Marmara University
Faculty of Engineering

CSE3063
Object-Oriented Software Design

Requirement Analysis Document

Instructor: Murat Can GANİZ Date:07.12.2023

	Department	Student Id Number	Name & Surname
1	CSE	150120012	Kadir BAT
2	CSE	150120055	Muhammed Talha KARAGÜL
3	CSE	150120020	Mustafa Said ÇANAK
4	CSE	150121520	Ensar Muhammet YOZGAT
5	CSE	150121076	Abdullah KAN
6	CSE	150120997	MOHAMAD NAEL AYOUBİ
7	CSE	150121021	Feyzullah ASILLIOĞLU

Vision

The vision of our course registration system is to establish a seamless and collaborative platform that enhances the efficiency of the course registration process within our university department. By integrating interactive features for students, advisors, and lecturers, the system aims to streamline course enrollment, provide comprehensive academic support, and foster effective communication among stakeholders. Our vision is to create a user-friendly environment that empowers students to manage their academic journey while enabling advisors to guide and assist with ease.

Scope

User Roles and Responsibilities

Students: Students will be able to enroll in courses, access their transcripts, consult weekly class schedules, and receive notifications related to the registration process. They can communicate with their advisors through messages, receiving guidance and updates.
Advisors: Advisors will guide and assist students, facilitate the course selection process, and access detailed student information. They can receive notifications about the registration process, communicate with students via messages, and maintain an effective dialogue for academic support.
Lecturers: Lecturers can view their schedules and the courses they are assigned to. While they can access essential information, they are restricted from enrolling in the course registration system.

Course Registration Process

1. The system will support an interactive and user-friendly course registration process for students, allowing them to select and enroll in courses based on availability, prerequisites, and advisor recommendations.
2. Advisors will have the capability to review and approve course selections, providing valuable input and ensuring that students make informed decisions aligned with their academic goals.

Communication Features

1. A messaging system will facilitate communication between students and advisors, enabling a seamless exchange of information, guidance, and updates.
2. Notifications will keep all stakeholders informed about key events, deadlines, and changes in the registration process.

Academic Information

1. Students can access their transcripts, providing a comprehensive overview of their academic progress.
2. Advisors will have detailed access to student information, allowing them to provide tailored guidance and support.

Glossary

- **Course:** A syllabus module offered by the university.
- **Elective Course:** A course that may be chosen from any other department. When added to mandatory courses, form the total credits needed to graduate.
- **Non-technical Elective:** A course whose area is not related to the technical or specialized courses of the department. It's given to let the student explore his interest in other areas.
- **Faculty Technical Elective:** A course whose area is strongly related to the technical or specialized courses of the department. It's given to let the student develop his skills in a particular field.
- **Non-Elective Course:** An essential, mandatory course in the department that students are required to take.
- **Prerequisite Course:** A course that must be completed by a student to be allowed to take a higher-level course.
- **Prerequisite Tree:** A hierarchical illustration that represents the required order of prerequisite courses, providing the relationships between courses.
- **User Interface:** Interactive labels students use to interact with the system.
- **Course Request:** A request sent by the student to his/her advisor to take a course at the beginning of the semester.
- **Weekly Schedule:** A timetable that provides the distribution of the courses over a week.
- **Credit Limit:** The maximum number of credits set by the university to take by a student for a semester.
- **Enrolment Capacity:** The maximum number of students to register for a course.
- **Semester:** An academic term in which students engage with a set of courses they picked.
- **Notification:** A mechanism for alerting users about important updates in the course registration system, such as course approval or denial notification.
- **Message:** An entity that is used in the system to store information about communication between users.
- **Check Permissibility:** Checking if a student is eligible or allowed to register for a course.
- **Actor:** A user that interacts with the system, which is a student, advisor, or lecturer in our case.
- **Encapsulation:** Hide private data of a class and the functions that operate on them from other programmers that will use the class.

- **Log:** Logs contain information such as timestamps, event descriptions, and other relevant data.
- **Advisor:** A lecturer who mentors many students, traces their performances, and approves the course requests sent by them for the upcoming semester.
- **Domain Model:** An illustration that shows conceptual real-world classes in the problem domain.
- **Use Cases:** A collection of success and failure scenarios that describes an actor using the system for a purpose.
- **JSON File:** A language-independent text file that contains readable and interchangeable data. These files include information for students and advisors.
- **Transcript:** A detailed record of the student that shows his/her completed courses.
- **Non-Functional Requirements:** Requirements that describe the attributes and properties of the system.
- **Functional Requirements:** Requirements that describe features and behaviors wanted in the system.
- **System Sequence Diagram:** An illustration that shows the events that external actors generate for a specific use case.
- **Polymorphism:** This means many forms that a form may execute a method of a super form without casting. Polymorphism is a way of executing and implementing an inherited method in a manner specific to the subclass.

Functional Requirements

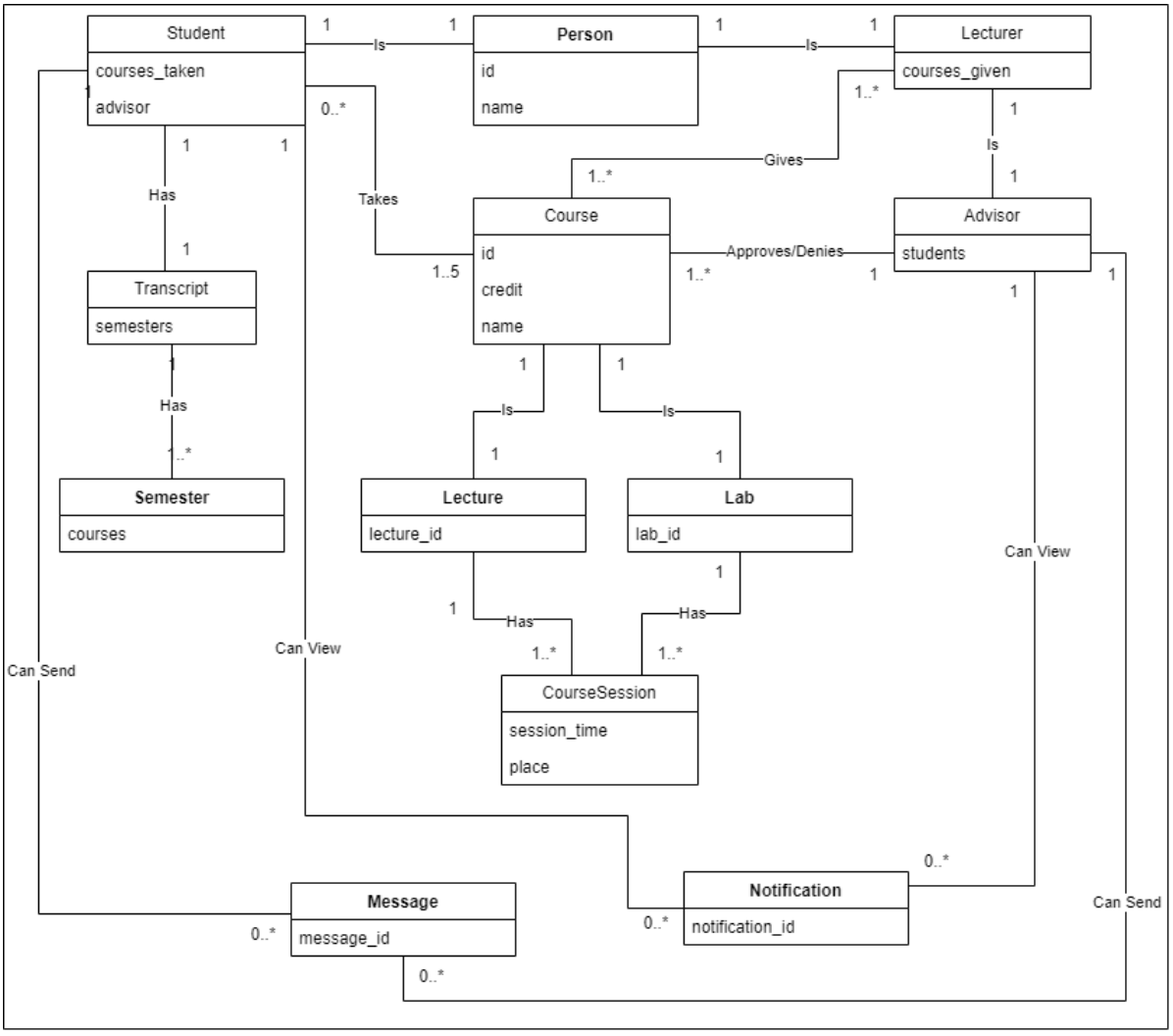
- Users who are Students, Lecturers Advisor's personal data should be kept in .json files.
- The user should log in with a pre-created unique user ID and password. Login information will be checked from the contents of the pre-created .json files.
- Logs of logging in, logging out and exits are saved to app.log file.
- For each student, the course registration process should be done based on availability and prerequisites and problems with handling registration should be checked.
- The system will provide the information on the prerequisite tree of courses to students based on their request to see the courses they can register for.
- Advisors can approve or deny courses and should decide the approval of their students' courses due to predefined university rules.
- After sending registration requests by students or finalization of course registration requests by advisors, are saved into logs which hold user activities .
- Each student should have a transcript kept as separate .json files for each student. Successfully registered courses will be recorded on the student’s transcript which will have both failed and passed courses.
- Students can view their transcripts by sending requests to the system. Also, they can see their syllabus which shows the course section information
- Students can see their weekly schedule after finalization of their registration process.
- Students and Advisors can view the notifications that are created by the system about the registration process.

- Students and Advisors can send messages and view messages that they have sent and received.
- Lecturers and Advisors can see their given courses and their weekly schedule.

Non-functional Requirements

- **Performance**: The login and course registration process should be quick and maintain efficiency with minimum delays.
- **Expandability**: As input data increases in the system, the program should allow a huge number of users without causing system failure.
- **Maintainability**: The program should be open for modifications and maintain any changes done to our code, which might help us as we move forward in the next iterations.
- **Security**: Users' passwords should be encrypted due to security measures.
- **Accessibility**: The program should have a user-friendly interface to avoid complications.
- **Consistency**: Reliability of the program is a must as data or case conditions change, the program should give consistent results.

Domain Model



Use Cases

Use Case Name: Registration for the Course

Summary:

Designed to enable students to successfully register for courses in the upcoming semester. This scenario involves interactions where students and advisors engage in the course selection, validation, and confirmation processes.

Actor:

Primary Actor:

- Student

Supporting Actor:

- Course Registration System

Stakeholders and Interests:

- **Student:** Wants to successfully register for courses for the upcoming semester.
- **Advisor:** Provides academic counseling to his/her students who is a lecturer. And he/she is the person responsible for his/her students' course selections.
- **Course Registration System:** Allows students to see the courses they will take and their syllabus based on the courses they will take.

Preconditions:

- The student must be logged into the system.
- A student has only one advisor.

Success Guarantee (Postconditions):

- The courses selected by the Students
- Students’ requests are saved into the JSON file
- The requests sent to the Advisor

Flow of Events:

Main Success Scenario (or Basic Flow):

Actor Action (or Intention)	System Responsibility
1. The Students enter the Course Registration System.	
	2. Retrieves student information from the database (JSON file)
3. The Student can see their notifications.	
	4. Retrieves student notification from the database (JSON file)
5. The Student can see weekly schedule	
	6. Retrieves data and calculates weekly schedule.
7. The Student can see their transcript	
	8. Retrieves student transcript from the database and calculates.
9. The Student can see the curriculum.	
	9. Retrieves curriculum from database..
10. The Students can see a list of all the courses he/she can take. **11.** The Students select the course they want. **12**. The Students can see a list of course sessions and labs they can take. **13.** The Students select the session and lab. The Student repeats steps 10-13 until he/she wants to save the selection. **14**. The Students save their course selection.	
	15. Saves information to the database (.json file) **9.** Place lessons in the syllabus
16. The Students can send messages to the Advisor.	
	17. Save messages to the database.
18. The Students can view their selected courses and total credits. **19**. The Students send their selection requests to the advisor.	**20.** The student registration process updated **21**. Notification sent to the Advisor **22.** Students request log saved to database (Log File).

Extensions (or Alternative Flows):

3a. If the student does not have any notification, return:

1. Go to main flow

11a. If there is an overflow in the credit system, give an error:

1. Go to step 10
2. Choose the courses that don't exceed the credit limit that can be taken

11b. If there is an overflow, the maximum number of courses is:

3. Go to step 10
4. Choose the courses that don't exceed the maximum number of courses that can be taken

11c. If the course you choose does not meet the prerequisites:

1. Go to step 10
2. Choose the courses that meet the prerequisites.

11d. If there is a conflict in the course section:

1. Give an error message to the Student
2. Go to step 10
3. Choose the courses that don’t conflict

11e. If the student wants to delete their selection:

1. Go to the deletion tab in the Course Registration Interface.
2. Prompt to user to which courses to be deleted

Special Requirements:

- The system must have up-to-date information on course availability, prerequisites, and enrollment capacity.

Technology and Data Variations List:

1-7a. Operations are done by keyboard.

Frequency of Occurrence:

- This use case occurs at the beginning of each semester when students need to register for courses.

Open Issues:

- How will the system handle problems where a course section reaches maximum capacity during registration?

Use Case Name: Course Registration Approve/Denied

Summary:

Crafted to facilitate advisors in efficiently guiding students through the course registration process for the upcoming semester. This scenario entails interactions where advisors and students collaboratively participate in course selection, validation, and confirmation procedures.

Actor:

Primary Actor:

- Advisor

Supporting Actor:

- Course Registration System

Stakeholders and Interests:

- **Student:** Wants to successfully register for courses for the upcoming semester.
- **Advisor:** Provides academic counseling to his/her students who is a lecturer. And he/she is the person responsible for his/her students' course selections.
- **Course Registration System:** Allows students to see the courses they will take and their syllabus based on the courses they will take.

Preconditions:

- The advisor must be logged into the system.
- An advisor must have at least one student.

Success Guarantee (Postconditions):

- The selected course information is sent to the Student
- Approve/denied saved into the JSON file
- The student's transcript is updated

Flow of Events:

Main Success Scenario (or Basic Flow):

Actor Action (or Intention)	System Responsibility
1. The Advisor enters the Course Registration System.	
	2. Retrieves students and requests information from the database (JSON file).
3. The Advisor can see their notifications.	
	4. Retrieves advisor notification from the database (JSON file)
5. The Advisor can see weekly schedule	
	6. Retrieves data and calculates weekly schedule.
7. The Advisor can see their given courses.	
	8. Retrieves advisor given courses from database.
9. The Advisor sees all students under his/her responsibility.	
10. The Advisor chooses a student.	
	11. Retrieve student's request from the database (JSON file).
12. The Advisor sees the courses requested by the student.	
13. The Advisor selects which courses to approve/deny. The Advisor repeats steps 10-13 until he/she wants to approve the request.	
	14. Saves approved/denied courses to the database (JSON file).
15. The Advisor finalizes the students request.	
	16. The student's course registration status is updated in the database (JSON file). **17**. Notification sent to the Student. **18.** Saves finalization log to the database (JSON file).

Extensions (or Alternative Flows):

3a. If student does not have any notification, return:

2. Go to main flow

13a. If there is an overflow in the credit system, give an error:

1. Go to step 12

2. Choose the courses that don't exceed the credit limit that can be taken

13b. If there is an overflow, the maximum number of courses is:

1. Go to step 12
2. Choose the courses that don't exceed the maximum number of courses that can be taken

13c. If the course you choose does not meet the prerequisites:

1. Go to step 2
2. Choose the courses that meet the prerequisites.

Special Requirements:

- The system must have up-to-date information on course availability, prerequisites, and enrollment capacity.

Technology and Data Variations List:

1-7a. Operations are done by keyboard.

Frequency of Occurrence:

- This use case occurs at the beginning of each semester when students need to register for courses.

Open Issues:

- How will the system handle problems where a course section reaches maximum capacity during registration?

Use Case Name: Lecturer

Summary:

During the semester, within the framework of the academic calendar, the lecturer has the opportunity to examine the course programs in detail. Additionally, through this system, they can view detailed information about the courses they teach.

Actor:

Primary Actor:

- Lecturer

Supporting Actor:

- Course Registration System

Stakeholders and Interests:

- **Lecturer:** Wants to see weekly class schedules and given courses
- **Course Registration System:** It allows lecturers to see the courses they will give and their weekly schedules according to the courses.

Preconditions:

- The lecturer must be logged into the system.

Success Guarantee (Postconditions):

- She/He will have seen which lesson he taught on which day.
- Will be able to see which course you teach in a semester

Flow of Events:

Main Success Scenario (or Basic Flow):

Actor Action (or Intention)	System Responsibility
1. The Lecturer enters the System.	
	2. Logging in of lecturer log entered to database (JSON File).
3. The Lecturer sees all the courses taught that term.	
	4. Retrieves the lessons given by the lecturer from the database(JSON file).
5. The Lecturer sees all the courses in the weekly schedule.	
	6. Retrieve the student's request from the database (JSON file).

Extensions (or Alternative Flows):

4a. If there is an error in viewing the given courses:

1. Go to step 1
2. Enters the System

4b. If there is an error in viewing the weekly schedule:

1. Go to step 1
2. Enters the System

Special Requirements:

- The system must have up-to-date information on course availability, prerequisites, and enrollment capacity.

Technology and Data Variations List:

1-5a. Operations are done by keyboard.

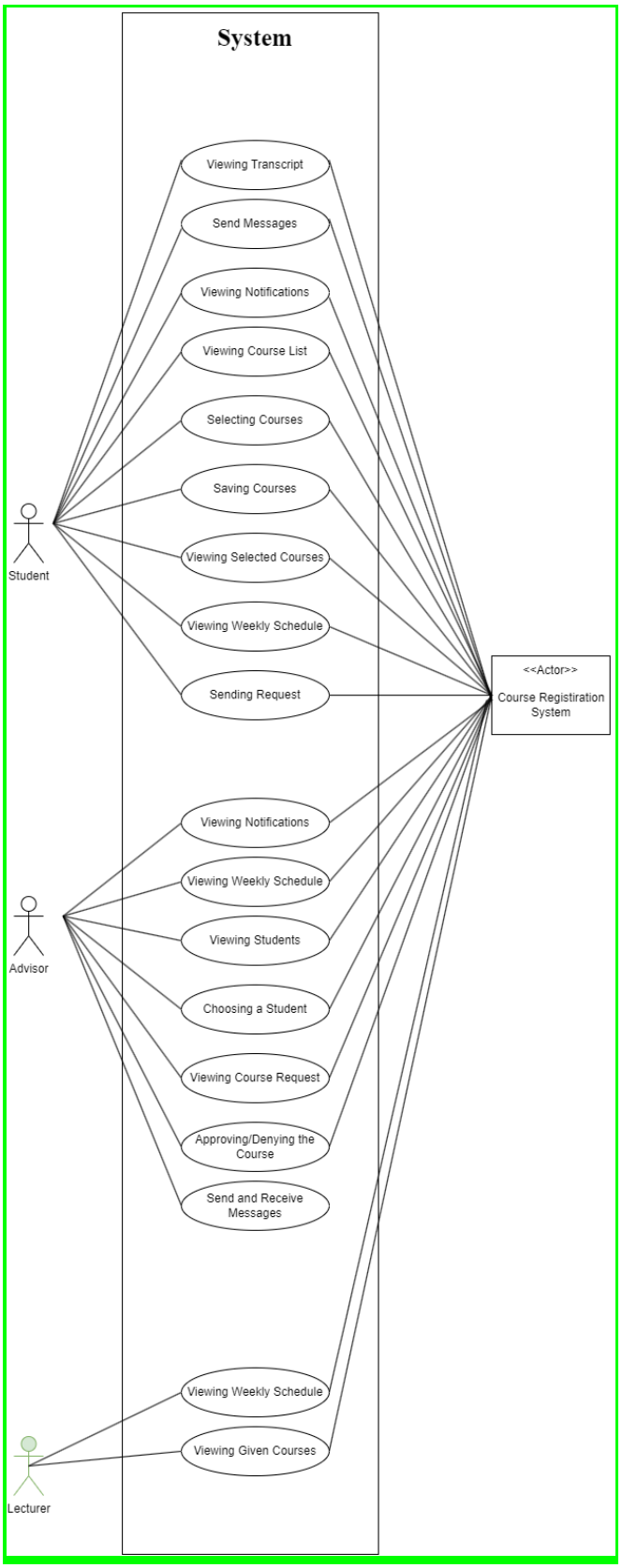
Frequency of Occurrence:

- This use case occurs in every term of the year.

Open Issues:

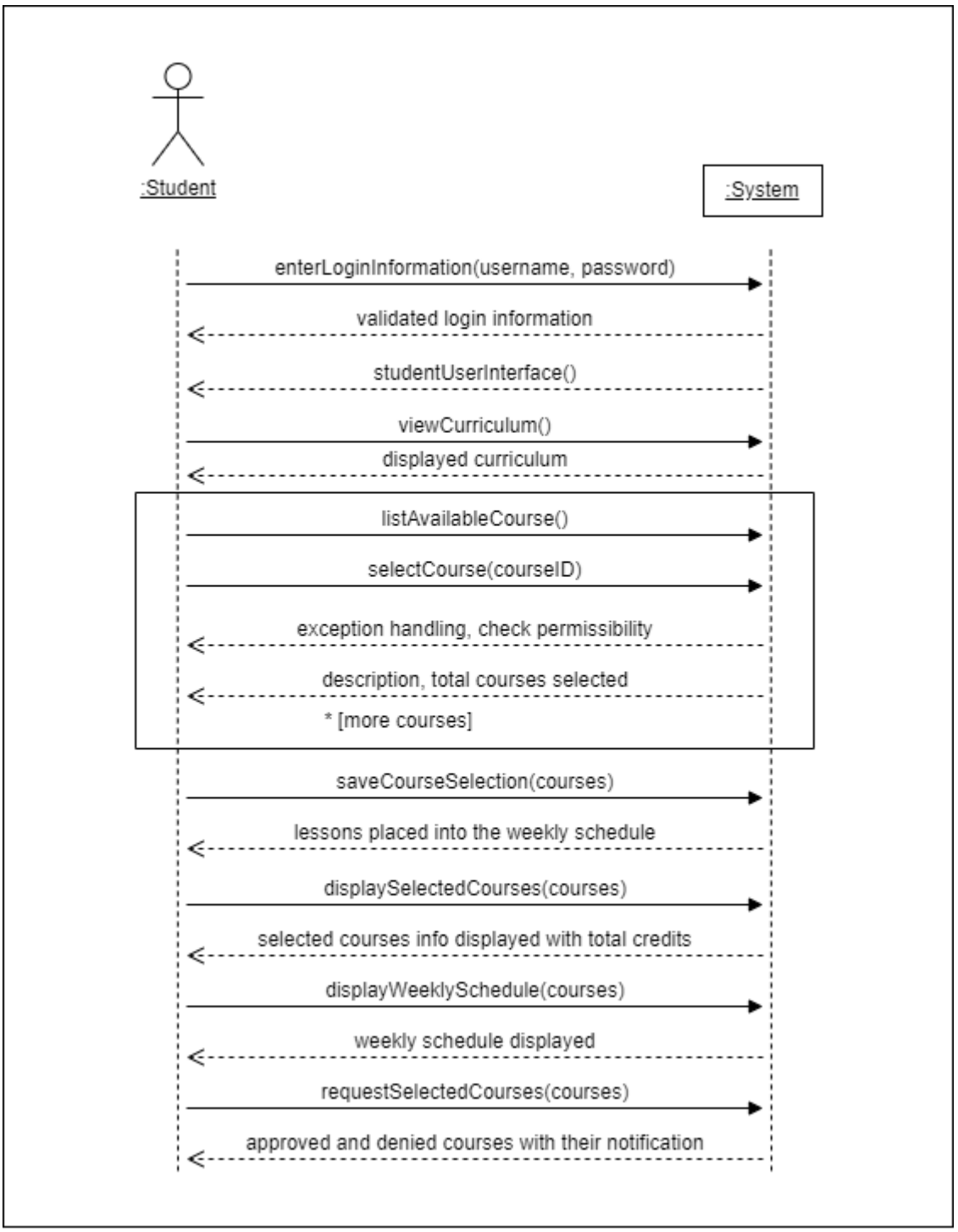
- How will the system produce a solution in case there are classes outside the set lesson times when displaying the weekly lesson schedule?

Use Case Diagram

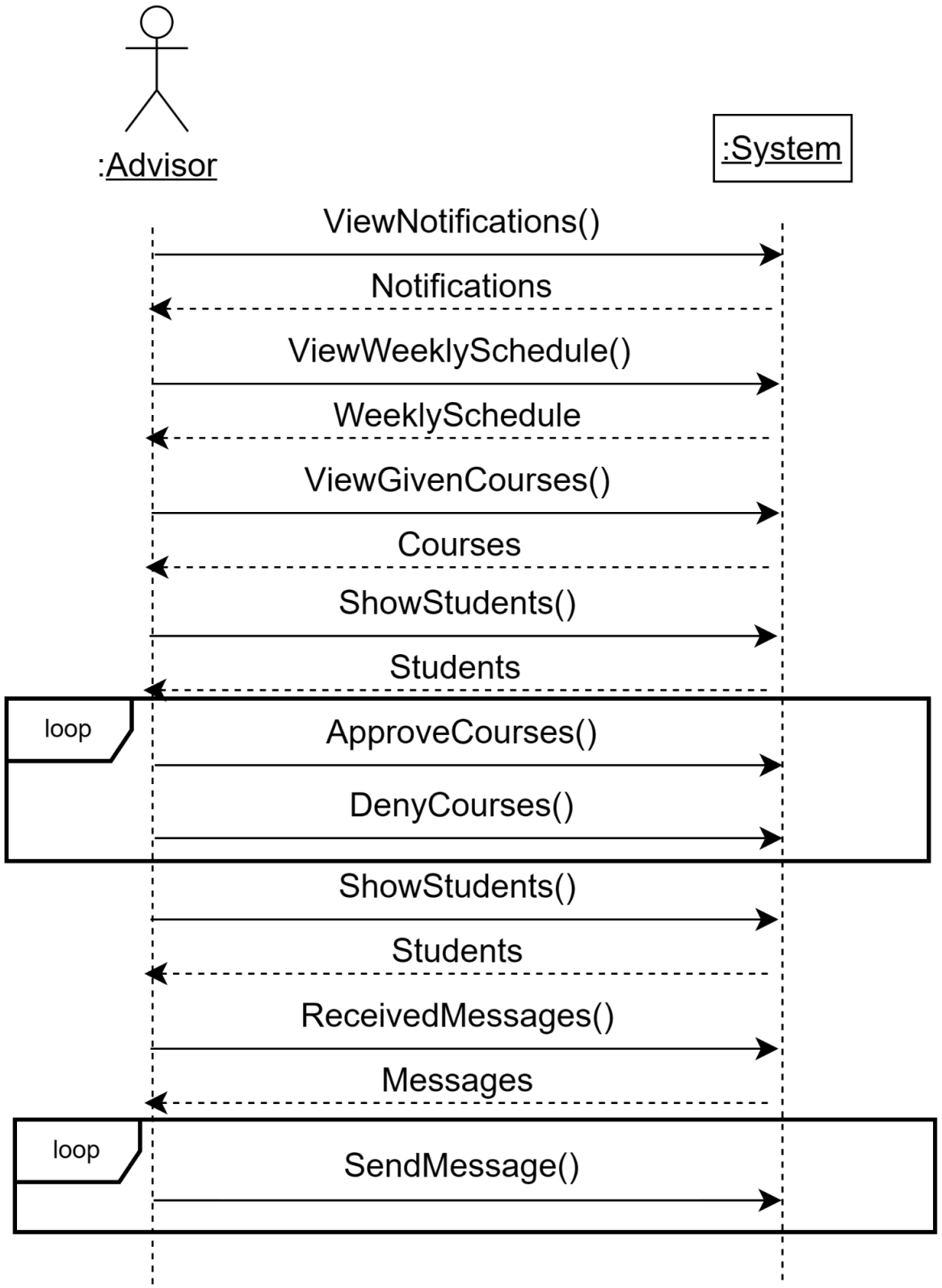


System Sequence Diagrams

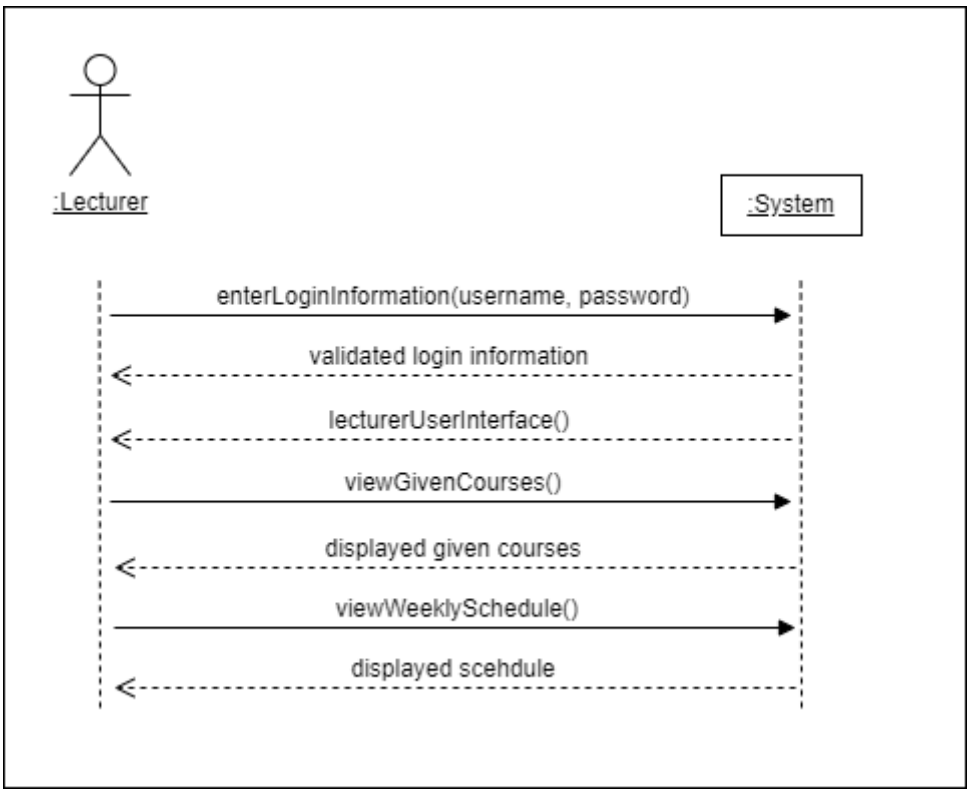
Student - System (SSD_1)



Advisor - System (SSD_2)



Lecturer - System (SSD_3)



Project Management

In each iteration, we have started our project by requirement analysis. First, we have discussed the functional requirements of the project with all group members. After having the features that we should design for the project, we created our domain model according to our functional requirements. Next, to have our dynamic design for the project, we have discussed the use cases for both students and advisors. Finally, we have split in design and remaining parts of the requirements analysis.

After completing software design implementations for our project, we have moved to implementation. We have defined different due dates for different classes to implement the design. Since some of the classes cannot be done before the required classes are done. So, we gave priorities to the classes to work efficiently.

Although we have made some tests during the implementation, we have also had another test session after all the implementation has been done by all group members to be sure everything works fine in each iteration.

Teamwork Distribution

Iteration 1

RAD

Abstract: Mustafa Said Çanak
Glossary: Mohamad Nael Ayoubi
Functional Requirements: Mustafa Said Çanak
Non-Functional Requirements: Kadir Bat
Domain Model: Feyzullah Asıllıoğlu
Use Cases: Ensar Muhammet Yozgat, Muhammed Talha Karagül
Use Case Diagram: Muhammed Talha Karagül
SSD 1 (Student): Mohamad Nael Ayoubi
SSD 2 (Advisor): Abdullah Kan

DCD

Mustafa Said Çanak
Kadir Bat
Mohamad Nael Ayoubi

DSD

Kadir Bat
Abdullah Kan
Muhammed Talha Karagül

Implementation

LoginInterface: Mustafa Said Çanak
StudentInterface: Kadir Bat
AdvisorInterface: Abdullah Kan
StudentCourseRegistrationInterface: Feyzullah Asıllıoğlu
AdvisorCourseRegistrationInterface: Ensar Muhammet Yozgat
Person: Mustafa Said Çanak
Student: Kadir Bat
Advisor: Abdullah Kan
Transcript: Mohamad Nael Ayoubi
Course: Muhammed Talha Karagül
Menu Functions Implementation: Muhammed Talha Karagül

Tests

Mustafa Said Çanak
Kadir Bat
Abdullah Kan
Feyzullah Asıllıoğlu
Ensar Muhammet Yozgat
Mohamad Nael Ayoubi
Muhammed Talha Karagül

UnitTests: Ensar Muhammet Yozgat

Iteration 2

RAD

Vision and Scope: Mustafa Said Çanak
Glossary: Mohamad Nael Ayoubi
Functional Requirements: Mustafa Said Çanak
Domain Model: Feyzullah Asıllıoğlu
Use Cases: Muhammed Talha Karagül, Ensar Muhammet Yozgat
SSD (Student and Lecturer): Mohamad Nael Ayoubi
SSD (Advisor): Abdullah Kan

DCD

Mustafa Said Çanak
Kadir Bat

DSD

Kadir Bat
Abdullah Kan

Implementation

Person: Mustafa Said Çanak
Student: Feyzullah Asıllıoğlu
Lecturer: Abdullah Kan
Advisor: Kadir Bat
Transcript: Mohamad Nael Ayoubi
Semester: Mohamad Nael Ayoubi
Course: Mohamad Nael Ayoubi
Lecture: Kadir Bat
Lab: Abdullah Kan
CourseSession: Muhammet Talha Karagül
CourseSessionTime: Muhammet Talha Karagül
WeeklySchedule: Muhammet Talha Karagül
Schedule: Kadir Bat
Colors: Muhammet Talha Karagül
Notification: Feyzullah Asıllıoğlu
Message: Abdullah Kan
Session: Mustafa Said Çanak
LoginInterface: Mustafa Said Çanak
StudentInterface: Kadir Bat
AdvisorInterface: Abdullah Kan
StudentCourseRegistrationInterface: Feyzullah Asıllıoğlu
AdvisorCourseRegistrationInterface: Ensar Muhammet Yozgat
NotificationsInterface: Ensar Muhammet Yozgat
MessagesInterface: Ensar Muhammet Yozgat

All the Menu and Print Functions: Muhammet Talha Karagül

Tests

Mustafa Said Çanak
Kadir Bat
Abdullah Kan
Feyzullah Asıllıoğlu
Ensar Muhammet Yozgat
Mohamad Nael Ayoubi
Muhammed Talha Karagül

UnitTests: Ensar Muhammet Yozgat