**Requirement Analysis Document**

Group Number: 9

Names:

-Yigit Tuncer (150121073)

-Hasan Pekedis (150120068)

-Ahmet Arda Nalbant (150121004)

-Hasan Özeren (150121036)

-Umut Bayar (150120043)

-Niyazi Ozan Ateş (150121991)

-Mehmet Sina Çağlar (150123821)

**DESCRIPTION**

The main objective of this project is to convert a Java project into a Python project. The goal of the project is to develop a functional course registration system based on our department. This system will be a user-friendly platform for the users, which are the students and staff of the university, to login with a user ID and password. After they have logged in, the system will implement a role-based access control to ensure that each type of user has access only to the features and data relevant to their role. It will provide an efficient catalog of courses for the students to send their course registrations and the advisors can manage this course registration by approving or denying the requests they receive. Also, the students will see which courses they can register according to the rules and regulations the department wishes. Beside this, they also get a view of their transcript, which will show their grades. The grades are controlled by the teacher, which also decide to pass or fail the student. To ensure the successful development of the system, close collaboration with the department and its stakeholders will be necessary, with detailed requirements and specific departmental rules clarified through consultation sessions, including classroom interactions, to make sure the system aligns perfectly with the department's needs. Exception handling and logging is a must part added to the system. While the system is created, we have given importance to future expanding such as making it adaptable to additional roles like department head, admin, etc. Beside this, the system aims to streamline the course registration process, increase efficiency, and provide a user-friendly environment for all users of the system, with room for potential future expansion and adaption.

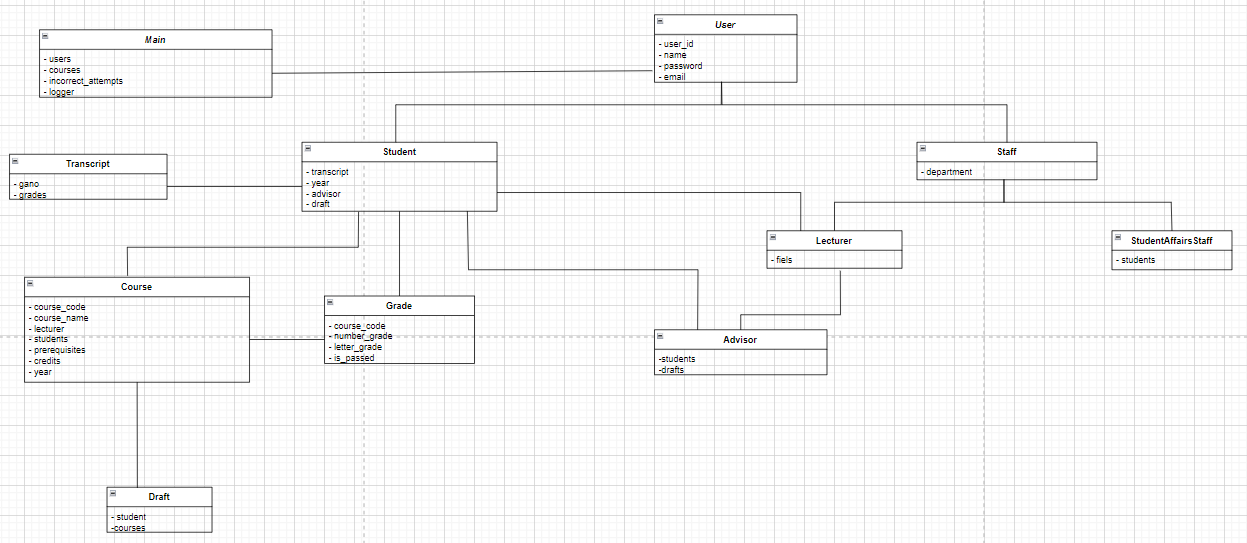
**GLOSSARY**

1. **User Management:** Concepts related to user registration, login, role assignment, and password reset functionalities.
2. **User Roles:** Allowing users to have different roles (e.g., student, advisor, administrator) and defining specific permissions for each role.
3. **Course Registration:** Understanding the process of users enrolling in courses for specific terms and the associated workflow.
4. **Courses and Programs:** Creating a database or data structure to store information about the courses offered by the department and academic programs.
5. **Registration Rules:** Department-specific rules, policies, and limitations that define how registrations can be made and under what conditions registrations are accepted.
6. **Approval Process:** Defining how advisors approve student registrations and the overall approval workflow.
7. **Course Selection and Cancellation:** Managing course selection, editing existing registrations, and cancelling registrations as needed.
8. **Performance and Scalability:** Optimizing system performance and understanding how the system can scale to handle high demands.
9. **Usability and Interface Design:** Designing a user-friendly interface that allows users to interact with the system easily.
10. **Error Handling and Logging:** Dealing with system errors, creating error logs, and monitoring error tracking.
11. **Database Design:** Determining how data related to courses, users, registrations, and rules will be stored in the database (JSON).
12. **Time Management:** Concepts related to terms, course durations, registration deadlines, and time-based processes.
13. **Stakeholders:** Individuals or groups with an interest or concern in the success of the software project.
14. **RAD (Rapid Application Development):** A software development methodology that prioritizes quick development cycles and iterative prototyping.
15. **Iterative Development:** A development approach that involves repeating cycles of prototyping, testing, and refining to progressively improve the software.
16. **Dependency:** A relationship between two tasks or components where the completion of one dependent on the other.
17. **Role-Based Access Control:** A security mechanism that restricts system access to authorized users based on their roles, ensuring each user type has appropriate access privileges.
18. **Catalog of Courses:** An organized and efficient list of available courses that students can browse and select for registration.

**LIST OF REQUIREMENTS**

|  |  |
| --- | --- |
| **Functional Requirements** | **Non-Functional Requirements** |
| Students, advisors, and lecturers should be able to log into the system with their assigned username and password. | Database operations should support. |
| Implement a Role-Based Access Control to ensure that different user types have access only to features and data relevant to their roles. | The system should be compatible with different devices. |
| Ensure an intuitive navigation system for easy interaction with the course registration system. | A user-friendly interface should be provided for users. |
| At least three roles should be defined for users in the system: Student, Advisor and Lecturer. It should be easy to add more roles for future requirements, such as Department Head, Admin, and Student Affairs. | Critical data should be regularly backed up to prevent data loss. |
| Students should be able to enroll in courses and advisors should be an able to approve students' course selections. | Design the system to be scalable, allowing for potential future expansion and adaptation to additional roles or features. |
| Courses and programs available in the system should be viewable and selectable during the registration process. | Conduct thorough testing, including unit testing, integration testing, and user acceptance testing, to ensure the system's reliability and functionality. |
| Department registration rules should be defined and enforced in the system. |  |
| Students should be able to select and register for courses. |  |
| Implement a transcript feature allowing students to view their academic history, including grades for completed courses. |  |
| Students may need to submit a registration application before making course registrations. |  |
| Advisors should have the authority to approve or disapprove students' course registrations. |  |
| Lecturers will have the ability to grade and pass/fail a student. |  |
| Include mechanisms for handling unexpected errors and exceptions to ensure system stability. |  |
| Implement logging to record system events for audit trails and debugging purposes. |  |

**DOMAIN MODEL**



**USE CASES**

**Use Case Name:** Enrolling classes

**Summary:** In order for students to select courses, they must first successfully log in to the system with their username and password, select their courses, save them as a draft and send them to their advisor.

**Subject:** Student

**Basic Flow:**

1) Student logs in to the website with their username and password.

2) Student opens "register to a course" tab to view courses.

3) The student selects courses from the course list. The course list is based on student’s current curriculum, current semester, and current course progress.

4) Student submits draft to their advisor for approval.

5) Student logs out.

**Alternative Flow:**

* Step 1: If student enters the username or password incorrectly three times, the system warns the student and gives a timeout, then use case returns to step 1.
* Step 2: If student opens “see the transcript” tab, the system shows students’ transcript, then use case returns to step 2.
* Step 3: If student enters the course incorrectly, the system warns the student, then use case returns to step 3.
* Step 4: If student submits incorrectly, the system warns the student and does not send it to the advisor, then use case returns to step 4.

**SYSTEM SEQUENCE DIAGRAM (SSD)**

1) Students log in with using their username and password if there are no conflicts.

2) Students enters the “register to a course" tab to view available courses.

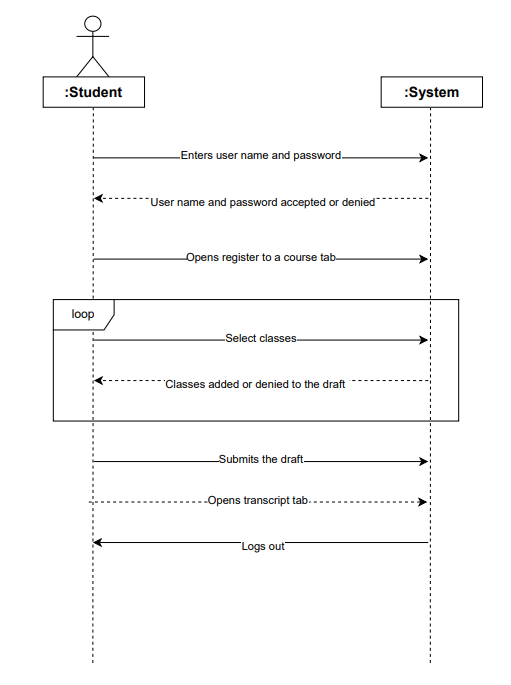
3) After selecting the courses, student submits the drafts for approval if there no conflicts (misspelling etc.).

4) If students open “transcript” tab, the system shows students’ transcript.

5) Students log out.

4) The advisor reviews the courses the student has selected.

5) The chosen courses have the advisor's approval.



**Use Case Name:** Approving/Rejecting Course Drafts

**Summary:** In order for students to be registered to the courses, the advisor must enter the system and approve or reject the requests from the students.

**Subject:** Advisor

**Basic Flow:**

1) The advisor logs in to the website with their username and password.

2) The advisor opens "student registrations" tab to view registrations.

3) The advisor evaluates the drafts chosen by the students and decides whether they will be accepted or rejected.

4) Advisor logs out.

**Alternative Flow:**

* Step 1: If advisor enters the username or password incorrectly three times, the system warns the advisor and gives a timeout, then use case returns to step 1.
* Step 2: If there is no drafts to approve currently, the systems gives a warning about nothing to see.

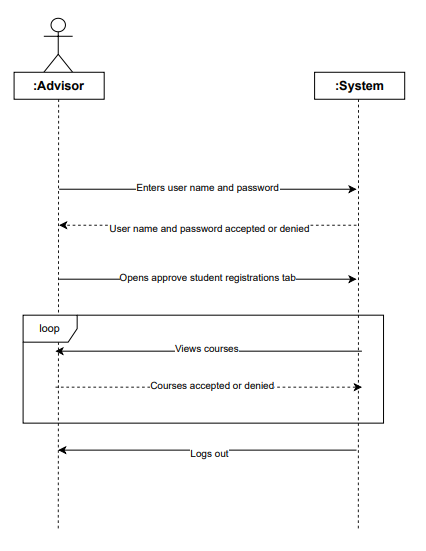
**SYSTEM SEQUENCE DIAGRAM (SSD)**

1) Advisors log in with using their username and password if there are no conflicts.

2) Advisors open "approve student registrations" tab to view registrations.

3) Advisors evaluate the drafts chosen by the students and decide whether they will be accepted or rejected.

4) Advisors log out.



**DESCRIBING TEAMWORK**

**Team Member’s Responsibilities**

* Yiğit Tuncer (150121073): Wrote a unit test for the student class. Implemented a year system as a prerequisite to taking classes. Updated and added features to the student class registration interface. Implemented course acceptance/rejection logic instead of the draft being completely accepted or rejected. Added JAVADOC comments to some of the classes, created a draft class instead of using a 2D array of courses. Implemented input sanitation to prevent runtime errors with taking input. The domain model documented for RAD document. For the third Iteration I built the login script and registration processor for Student. I also helped design the new DCD. I also helped with the rest of the process when needed. I also made the unit test for the student class.
* Hasan Pekedis (150120068): Implemented JSONFileManager class for JSON file manuplation and data management, created arrays for all componenets. All student, advisor, lecturer, student affairs staff and course JSON files are designed and samples created. LoggerSystem class created and tested for event tracking and debugging, logger messages added in LoginSystem. Timer implementation completed, used TimeUnit library to implement. JUnit test created for the Lecturer class and passed all tests. List of requirements and important concepts(glossary) documented for RAD document.
* Ahmet Arda Nalbant (150121004): Wrote a unit test for Advisor class with using JUnit library and passed all test. Wrote the Use Cases (one for student and one for advisor) for Iteration 1 and updated it for Iteration 2 and Iteration 3. Draw the System Sequence Diagram (SSD) for Iteration 1 and updated it for Iteration 2 and Iteration 3 with using drawio. Implemented advisor approval or rejection function to the code for Iteration 1. Implemented a new feature that when a user enters the password or user name incorrectly for many times, it gives a timeout (like in BYS). For the third iteration, timeout feature updated for pyhton, as well as the advisor approval or rejection function.
* Hasan Özeren (150121036): In the first iteration, I created the course class as code and enabled the selection of courses through this class and wrote a unit test for the course class. In addition, I created and prepared the DSD document in the first iteration. In the second iteration, I created the prerequisite logic for the courses and ensured that the courses were selected accordingly. In addition, I prepared 2 DSDs for Advisor and Student. For the 3rd iteration, I set up an eligable course in the code and edited it. In addition, I wrote 2 unit tests for grade and draft. I rearranged the DSD and prepared 2 design sequence diagrams for advisor and student in accordance with the code.
* Umut Bayar (150120043): Added new methods to the Student Affairs Staff class, which was one of the two courses wroted in the first iteration, and the other class was the staff class. Added all cases related to student affairs staff with the methods added to the Login system these are information updating and various information viewing. Wrote a unit test for the Staff class using the JUnit library for two iterations and passed all test. Wrote the Design Class Diagram for Iteration 1 and updated it for Iteration 2. In Iteration 3, the feature of viewing registered courses and updating student number was added to the student affairs staff section in the main.py class. Prepared DCD for Iteration 3.
* Niyazi Ozan Ateş (150121991): In the first iteration I have created the LoginSystem class. The purpose of this class is to give each person the functionality to login and provide the actions they can take, which my team-members have created. Also, I have created the classes, attributes, and methods for the DCD. Where I did this also in the second iteration. In the second iteration beside doing the same for the DCD, I have created the CourseInstance class which connects the students with the lecturers. Also, I have gave the lecturers the option to control their courses. By choosing a course, the lecturers can get information about the courses and the students that take that course, grade any student, and pass/fail any student for a particular course. Also, to help with the prerequisite, I added the isCompleted variable to the Course class that says if a student has completed that course. For the third iteration, I have created the lecturer part of the project from JAVA to PYTHON, which I have also done in the previous iterations. This include getting information about which courses a lecturer gives and grade a student. By grading a student, it should also pass/fail that student based on the rules and regulation made in other parts of the code. Also, I have created a method in the course class and I have made a unit test for the lecturer called TestLecturer. Beside these contributions, I have updated the description, glossary, and list of requirements part in the RAD documentation.
* Mehmet Sina Çağlar (150123821): In the documentation process, I played a role in crafting relationship arrows, including association, aggregation, composition, and inheritance for the UML class diagram. In the subsequent iteration, I have updated the domain model for the ultimate project. With the aim of demonstrating the implementation of polymorphism, I have introduced an abstract method within the User.java class. Additionally, I introduced the Transcript.java class, designed to provide detailed information about the student. Furthermore, I implemented the Grade.java class that helps for converting between various grade types seamlessly, which I made sure by implementing a unit test for this class.

**The Project Management**

We used a Kanban board to organize tasks. Each new idea, feature or bug fix got turned into a git issue and assigned to a project member. We have 5 columns on our Kanban board to organize tasks. To-do column for keeping track of unassigned issues, assigned for issues assigned to a team member, a In Progress column to show what who is working on at the current moment, a review column to show issues that are complete but should be reviewed by other team members before completion, and finally a done section for issues that are complete. While working on the project we sometimes worked on diverging features that required us to branch the project and then one we had a working feature or change, merge into the main branch. Sometimes we had to help each other out on some features.

**The Tools**

Kanban board

