Manage Project Homework: Requirement Analysis and Specification Document

Riccardo Ancona - 782025
Alessandro Ditta - 781482

January 24, 2012

Revision Summary

1. Date: 23/01/2012

   Revision: 1

   Changes:

   Modified the Registration of a new professor and definition of a new projectscenario.

   Modified the Start/Stop the system Use Case.

   Inserted the Customer specifications' unclear points subsection of the Specific Requirements section.

Contents

List of Figures

1 Introduction

1.1 Purpose

The purpose of this document is to describe the requirements specifications for a application used by professors and students of university courses with project deliverables, including the details of the project requirements, interfaces, design issues and components.

The intended audience of this document includes the prospective developers of the tool, the students, the professors, the system administrators who will use the framework and the people responsible for the course organization.

1.2 Scope

The software system to be produced is a projects management tool which will be referred to as Manage Project Homework (MPH) throughout this document.

MPH will allow professors to publish a project description and to define the set of the corresponding deliverables. It will allow students to join project teams and submit deliverables by uploading them into the system.

The professor will also be able to evaluate the project deliverables assigning a score to them and leaving to the system the computation of the final score based on the average of the individual scores. MPH will also provide some information sharing functionalities among different teams.

1.3 Definitions, acronyms, and abbreviations

MPH: Manage Project Homework, the software system to be produced.

Team: a set of 1, 2 or 3 students who work together on the same project.

Admin: the system administrator

Deliverable: the model of a tangible object produced as a result of a specific phase of the project.

Artifact: the effectively tangible object associated to a deliverable

Deadline: the date by which the artifact associated to a deliverable must be delivered

1.4 References

- Description of the project: *http://corsi.metid.polimi.it*
- IEEE Std. 830-1998: IEEE Recommended Practice for Software Requirements Specifications: *http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574*
- Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, Ingegneria del Software - Fondamenti e Prinicipi, Pearson Education Italia, 2004

1.5 Overview

The rest of this document contains a description of the MPH software system (Section 2) and specific requirements for the system (Section 3). Section 4 will present a model of the MPH software, Section 5 will show the system specifications.

2 Overall Description

2.1 Product perspective

In some university courses students are required to develop a project. A project is commonly structured in several phases, each phase involves the release of an artifact before a certain deadline assigned to each phase. To improve the management of the deliverables students and professors could use a software tool that automates the process of submitting and evaluating each artifact requested by the professor and provided by the students. Without the support of the tool the time spent on each of the previous activities can be a tedious task: MPH is meant to ease this work.

2.1.1 System Interfaces

The MPH system to be developed consists in a Client-Server application integrated with a database accessible by the Internet. It consists in three major components:

- the Database that contains the users' profiles' information (first name, last name, ID, etc.) and the project details (deadlines, scores, submitted deliverables, etc.)
- the Client Module which allows the users to use the system (register and log into the platform, creating/joining a new project, submitting/viewing deliverables, etc.)
- the Server Module is a daemon (a software running in background mode) which accepts connections from Client Modules and serves as an interface between the Module and the Database

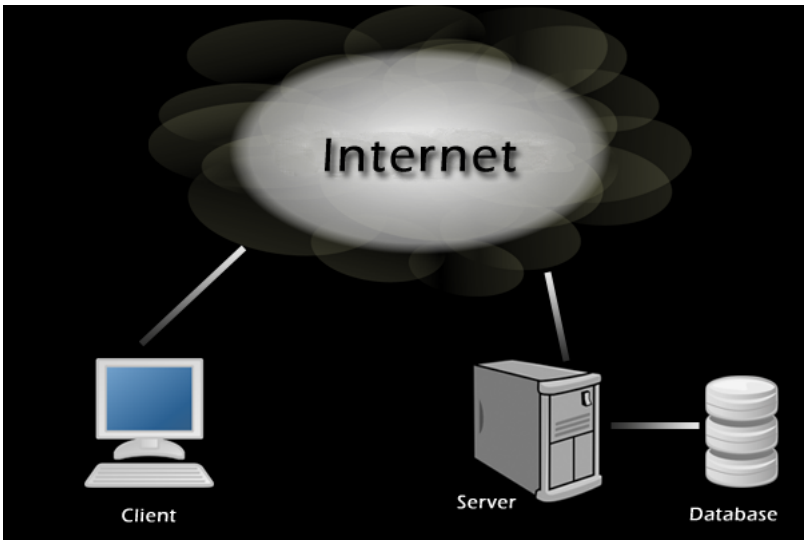

Figure 1: System Architecture

2.1.2 User Interfaces

The Client module must provide a user interface that is available through a Java application runnable on the main operating systems (Windows, Linux, Mac OSX). The Server Module has a minimal user interface that will allow the system administrator to manage users, permissions and to perform ordinary maintenance. The Database Module does not have a user interface.

2.1.3 Hardware Interfaces

All components must be executable on a personal computer.

2.1.4 Software Interfaces

MPH will be executed on a Java Virtual Machine (JVM). The proper behavior and visualization of the application is ensured on the following operating systems: Windows, Linux, Mac OSX.

2.1.5 Communication Interfaces

The Client Module must communicate with the Server Module with a TCP/IP connection. The Server and Database components must be located on the same host.

2.1.6 Memory Constraints

There are no specific memory constraints for MPH.

2.1.7 Operations

The operations of the Client Module must be easy and intuitive for students and professors. No specific formation must be required to use the tool. The Server Module must be installed and maintained with no interactions with other existing software and should not require any specific technical skill from a network administrator.

2.1.8 Site Adaptation Requirements

No specific site adaptation is required.

2.2 User characteristics

Users are professors and students enrolled in a university course with project deliverables. They are familiar with the the use of a personal computer and managing files, they are not required to know any additional technical skill. System Administrators must be familiar with the SQL language and have a good knowledge about databases and their maintenance.

2.3 Constraints

MPH must be developed using the JEE (Java Enterprise Edition) platform and, in particular, EJBs (Enterprise Java Beans) for the business logic. The system should enforce user authentication security and ensure deliverables' storage consistency.

2.4 Assumptions and dependencies

A Java Virtual Machine must be installed on the host running the MPH application (both Client and Server Module).

3 Specific Requirements

3.1 Customer specifications' unclear points

During the analysis of the software requirements, there were some unclear points in the customer specifications for the MPH software. Here follows a list of unclear points and their final resolution.

3.1.1 Deadline data format

From the specifications, is not known what kind of data format must have the deadlines associated to the deliverables. It was decided to use a date with the following format: *day-month-year*. A deadline expires at midnight of the specified day-month-year date.

3.1.2 Adding a new course

From the specifications, is not known if a professor can insert a new course into the MPH software or is a system administrator duty. It was decided to let ONLY the system administrator add a new course held by a professor.

3.1.3 Artifact default score

From the specifications, is not known what score must have a Deliverable when inserted into the MPH system. It was decided to set the default score of an Artifact to 1.

3.2 Scenarios

3.2.1 Registration of a new professor and definition of a new project

The professor Ditta wants to define a project for his own university course "Databases 2". To do so, he intends to use the software MPH, that he has never used yet. Professor Ditta contacts engineer Ancona, the system administrator of MPH, and informs him about his intention to use the software MPH. Ancona launches the MPH server module. After some time spent loading all the components, Ancona loads the MPH Database Software Manager, then he opens the "Execute SQL Query" window and types a query that creates a new professor, with a new professor ID, and the other professor's data like first name, last name, phone number and email address. After the creation of the new professor, Ancona types another SQL query that creates a new course called "Databases 2", held by professor Ditta. The engineer Ancona tells Ditta that he is now ready to use the software MPH. The professor Ditta logs into the system with the username and password chosen before and he accesses his course. He fills a project creation form, defining for the project its name, the start date, the default penalty constant for late deliverables and he sets the deadlines (with their dates). For each deadline, he defines a title, a date by which a deliverable must be uploaded and an optional custom penalty constant for late deliverables.

3.2.2 A student creates a team and another student joins it

The student Ancona wants to consult the list of projects that were started by professor Ditta within the university course "Software Engineering" and he wants to create a new working team for the "Dinosaurs Island" project. To do so, he enters the MPH software and invokes the registration form. He enters his username and password and confirms his intention to register to the MPH system. The system informs him that the registration has been performed successfully. Ancona then performs a search of the course "Software Engineering" and selects the only result. He can view all the currently active projects for the course. After selecting the desired item, in the new view that appears (the "Dinosaurs Island" project view) he clicks on "Create new project team". The system confirms the successful creation of the project team. Meanwhile, the

student Lo Bianco wants to consult the list of projects that were activated by professor Ditta within his university course and enroll in the project "Dinosaurs Island". To do so, he enters the MPH software and invokes the registration form. He enters his username and password and confirms his intention to register to the MPH system. The system informs him that the registration has been performed successfully. The student Lo Bianco then performs the same operation done by the student Ancona to access the "Dinosaur Island" project view. He can see a list of the teams associated to the project. After selecting the Ancona's team, he confirms his will to join that team. The student Ancona, who already joined the team, receives a team join request by Lo Bianco and accepts the request. When the student Lo Bianco logs into the system, he discovers to be part of the team.

3.2.3 A student upload an artifact for a deliverable

The student Ditta wants to send the "ProjectPlanning.zip" artifact for the deliverable associated to the deadline "Team Registration and Project Planning" of project "Software Engineering 2" held by prof. Ancona. To do so, he logs into the system, searches for his project team, he enters the "Deadlines" section, and then opens the form where he can upload the desired deliverable. He starts the upload process, and after a few seconds the system tells him that the artifact has been sent successfully.

3.2.4 A student leaves a team in order to develop the project alone

The student Cabrini wants to leave a team that she had previously joined, and before the project starts he wants to create a new team and carry out the project alone. To do this, she logs into the system, she enters the team view and selects the "Leave team" button. She confirms the intention to leave the team, and then returns to the main screen of the the project. Then she clicks on "Create new project team", and after confirming the choice and waiting a few seconds, the system confirms the successful creation of the new team for the project. Then she decides to block the membership requests to her team by unchecking the "Accept new membership requests" box in the main project window.

3.2.5 A professor evaluates some deliverables submitted by students

The professor Ditta wants to evaluate some deliverables sent by his students for the project "Database Design" of his course "Databases 2". He enters the main screen of the software, and accesses the course "Databases 2". Then he clicks on the project "Database Design". He takes a look at the list of deadlines of the project and the teams assigned to it. By clicking on a deadline, the professor access to all the artifacts sent by each team for that deliverable and the professor can view and download them to express his opinion about the delivered files. Next to each artifact there is a form that he can use to evaluate it; if a deliverable has not been delivered yet, its score is set to 1. The system

displays the final score for each deliverable calculated by subtracting the actual vote of the professor with a corrective penalty in case of late deliveries. Then the professor returns to the main screen and selects a team. He can see all the deliverables submitted by the team up to that moment, and the final score obtained by the team automatically calculated by the system.

3.2.6 The system administrator sets a new password for a professor who forgot it

The professor Ancona can not enter the MPH system because he forgot his password. He calls engineer Cabrini, the system administrator, in order to make him change his current password. Cabrini launches the MPH server module, she opens the MPH Database Software Manager, she opens up the "Execute SQL Query" view and write a query that let her change the password with a custom password, that she will tell to professor Ancona. Professor Ancona then logs into the MPH system and finally changes his password with a new password.

3.2.7 Example of project team creation (with leaving a team)

Two students, Ancona and Ditta, want to participate in a project of the course "Software Engineering 2" in the same team. Each of them registers in the MPH system, and update his own basic information. For example, after logging into the system and entering the profile section, Ancona enters his first and last name, phone number and email address. Then he decides to change his password: in the same section he enters his old password and the new password twice, for security reasons. After entering all the information, he confirms that his intention to change the password. The system verifies that the two new passwords are equivalent and that the old password is correct, then the operation ends successfully. After entering his personal data, Ancona decides to create the team. Ancona selects from the main screen the course, and then the project of interest, then he opens the form for creating a new team, he enters "MPH-Ancona-Ditta" as project name and confirms his choice. Ancona tells Ditta that the team has been created. Ditta logs into MPH, and from the main menu, he searches for the project "Ancona-Ditta", then he selects the button "Join Project" and confirms his decision. From the main page of the team, Ancona sees a "Join Team" request, and decides to accept it. Ancona then decides that the team is complete, and from the main screen of the project he unchecks the "Accept new membership requests" box. After some time, Ditta decides to leave the team before the start of the project: after logging into the system and reaching the team's main page, he clicks on "Leave team" and confirms his request. At this point Ancona decides to continue alone, not allowing other team membership requests, and he changes the team name to "MPH-Ancona". Another student, Cabrini, sees in the list of project teams the "MPH-Ancona" team, and she tries to join it: the system warns Cabrini that the team does not accept any new member, and Cabrini is forced to join another team. After the project has started, Ancona notices that he can not make any change to the

team composition nor changing the team's name.

3.2.8 Complex example of interaction between professors and teams

Professor Ancona has recently started his project "Game Making", and he must evaluate the artifacts produced by 2 teams, whose names are "Mushrooms" and "Codemasters". The student Cabrini of team "Mushrooms" wants to send a deliverable for the first deadline, "Project Planning". She logs into the system, enters the team main page, and then she clicks on the deadline and press "Upload Artifact". She chooses a file to send and confirms the upload. The student Zuech belonging to team "Codemasters" does the same, he sends his artifact for the deadline "Project Planning". The day after the deadline, professor Ancona wants to evaluate the work of the two teams. Ancona accesses to the system and opens the main page of the project. He accesses the deliverables view window, and downloads the two files delivered by the two teams by clicking the "Download" button next to each deliverable. After reviewing them, he decides to give them a score, so he inserts 10 and 4 respectively for the team "Mushrooms" and "Codemasters" in the "Proposed Score" box next to each artifact. The next day, two days after the "Project Planning" deadline, the student Zuech edits his artifact and resubmit his material for the same deliverable, running the above procedure once more. When he sends the file, the system informs Zuech that he is going to insert a late deliverable, and that the previous artifact will be lost. Zuech decides to continue with the operation, knowing that the final score will be reduced by 2 points since the penalty constant was equal to 1 and the days of delay are 2. After a while, professor Ancona realizes that there is a new late artifact for the "Project Planning" deliverable. He takes a look at it, and he evaluates with a score of 10.Although the system recognizes the artifact as late, and it subtracts a penalty equal to the constant penalty for the number of days of delay, so it shows 8 as "Final Score".

3.3 Use Cases

3.3.1 System Administrator

Register a New Professor

- *Participating Actors:* System Administrator
- *Pre-condition:* the Administrator wants to register a professor into the MPH system and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that creates a new professor, with his/her username, password and professor's e-mail

4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched.
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data and notices the successful registration of the new professor into the system

- *Post-condition:* the professor is correctly registered in the MPH system
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted
  - If the chosen username is not available, the Admin has to ask a new username to the professor and restart the registration procedure

Create a New Course

- *Participating Actors:* System Administrator
- *Pre-condition:* the Administrator wants to create a new course held by a professor into the MPH system and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that creates a new course held by a particular professor
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data and notices the successful creation of the new course into the system

- *Post-condition:* the new course is correctly created in the MPH system
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted
  - If the name of the course is not available, the Admin has to ask a new course name to the professor and restart the creation procedure

Edit User Profile

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to edit a user profile and has already launched MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that edits the desired user profile
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data and notices that the changes made to the user profile have been saved successfully

- *Post-condition:* the changes made to the user profile have been saved successfully
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted

Delete Users

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to delete certain users from the MPH system and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that will delete certain users from the MPH system
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched

6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data and notices that the desired users have been successfully deleted from the system

- *Post-condition:* the desired users were successfully deleted from the system
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted

Delete Course

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to delete a course from the MPH system and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that will delete a course from the MPH system
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data and notices that the desired course has been successfully deleted from the system
9. The Admin opens the directory containing the delivered artifacts and deletes it

- *Post-condition:* the desired course was successfully deleted from the system, as the teams associated to it and the artifacts uploaded
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted

View Table

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to view a particular table of the database and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that will let him to view the desired table
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data

- *Post-condition:* the Admin has viewed the desired table of the database
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted

Edit Delivered File's Metadata

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to edit a particular delivered file's metadata and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that will let him edit a particular delivered file's metadata
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results

7. MPH shows the results
8. The Admin observes the data and notices the changes made to the delivered file's metadata have been saved successfully

- *Post-condition:* the changes made to the delivered file's metadata have been saved successfully
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted

Delete Delivered Files

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to delete a particular delivered file and has already launched the MPH server module
- *Event flow:*

1. the System Administrator tries to open the "Execute SQL query" view
2. MPH shows the view correctly
3. the Admin types a query that will let him know the file system path of a particular delivered file
4. the Admin confirms the will to run the query
5. MPH notifies the Admin that the query was successfully launched
6. the Admin types another query to check the results
7. MPH shows the results
8. The Admin observes the data, and opens the desired directory
9. The Admin removes the desired file

- *Post-condition:* the delivered file is removed successfully from the system
- *Exceptions:*
  - If the SQL query is mistyped, the system won't execute it and the process has to be restarted
  - If in the directory there is no such file, the task is considered accomplished

Check System Log

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator wants to view the system log and the system is already started
- *Event flow:*

1. the System Administrator opens the System Log view
2. MPH shows the Admin the MPH system log, which contains all the actions performed by the server
3. the Admin filters the log view with the desired criteria by filling a filter form

- *Post-condition:* MPH shows the Admin a list of system actions corresponding to the desired criteria
- *Exceptions:*
  - If the fields of the filter form are not filled correctly, the Admin has to repeat the operation

Start/Stop the system

- *Participating Actors:* System Administrator
- *Pre-condition:* the System Administrator accesses the MPH system
- *Event flow:*

1. the System Administrator run the terminal
2. the Admin executes the commands to start or stop the MPH system
3. MPH shows diagnostic messages that inform the administrator that the server is running or is stopping
4. the system status is saved permanently
5. MPH notifies the Admin that the system has started or has been stopped correctly

- *Post-condition:* the system has started or has been stopped correctly
- *Exceptions:*
  - If MPH is not running, the Admin can not execute the Stop command
  - If MPH is already running, the Admin can not execute the Start command

3.3.2 Student

Register

- *Participating Actors:* Student
- *Pre-condition:* a Student accesses the MPH system
- *Event flow:*

1. the Student decides to register into the MPH system
2. MPH shows a registration form
3. the student chooses a username and a password
4. the student fills the form with personal information
5. the student confirms his/her registration
6. MPH notifies the student about the successful registration

- *Post-condition:* the Student is registered into the MPH system
- *Exceptions:*
  - If the registration form is not filled correctly by the student, the process has to restart
  - If the chosen username is not available, the student has to change it

Log In

- *Participating actors:* Student
- *Pre-condition:* the Student, who is already registered into the system, accesses MPH
- *Event flow:*

1. the Student executes the login command
2. MPH shows a login form
3. the Student fills the form with his/her username and password
4. MPH checks username and password and authenticates the Student

- *Post-condition:* the Student is logged into the system
- *Exceptions:*
  - If the inserted credentials are incorrect, MPH notifies the Student that he/she has to repeat the procedure

Update Profile Information

- *Participating actors:* Student
- *Pre-condition:* the Student is logged into the system
- *Event flow:*

1. the Student executes the command to edit his/her own profile information
2. MPH shows the Student his/her profile information page
3. the Student updates his/her personal details
4. the Student saves the changes
5. MPH notifies the Student that his/her data have been saved correctly

- Post-condition: the system updates the Student's profile data saved in the database
- *Exceptions:*
  - If the Student fills his/her profile page with incorrect data, MPH does not save any changes and the Student has to repeat the operation

Search for Students

- *Participating actors*: Student
- *Pre-condition:* the Student is logged into the system
- *Event flow:*

1. the Student executes the command to search for information about another student
2. MPH shows a search form
3. the Student fills the form with the username or the name of the desired student
4. MPH shows the Student a list of students corresponding to the desired criteria
5. the Student selects a student from the list

- *Post-condition:* MPH shows the Student the information about the selected student profile page
- *Exceptions:*
  - If the key fields of the search form are not filled correctly, the Student has to repeat the operation
  - If MPH does not find any entity matching the search form data, it notifies the Student, who can perform a new search

Search for Professors

- *Participating actors*: Student
- *Pre-condition:* the Student is logged into the system
- *Event flow:*

1. the Student executes the command to search for a professor
2. MPH shows a search form
3. the Student fills the form with the username or the name of the desired professor
4. MPH shows the Student a list of professors corresponding to the desired criteria
5. the Student selects a professor from the list

- *Post-condition:* MPH shows the Student the information about the selected professor and the list of courses held by the professor
- *Exceptions:*
  - If the key fields of the search form are not filled correctly, the Student has to repeat the operation
  - If MPH does not find any entity matching the search form data, it notifies the Student, who can perform a new search

Search for Courses

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing a list of professors
- *Event flow:*

1. the Student selects a professor from the list
2. MPH shows the Student a list of courses held by the professor
3. the Student selects a course from the list

- *Post-condition:* MPH shows the Student the information about the selected course
- *Exceptions:*
  - If the selected professor does not hold any courses, MPH notifies the Student, who can select another professor

Search for Projects

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing a list of courses
- *Event flow:*

1. the Student selects a course from the list
2. MPH shows the Student a list of projects corresponding to the selected course
3. the Student selects a project from the list

- *Post-condition:* MPH shows the Student a page containing the project description, the corresponding deliverables' deadlines and the project teams
- *Exceptions:*
  - If the are no projects corresponding to the selected course, MPH notifies the Student, who can select another course

Create Project Team

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing information about a project
- *Event flow:*

1. the Student executes the command to create a new project team
2. MPH shows a project team creation form
3. the Student fills the form with the name and description of the team he wants to create
4. the Student saves the changes
5. MPH notifies the student the successful creation of the new project team

- *Post-condition:* the project team is created into the system
- *Exceptions:*
  - If the Student fills the form incorrectly, MPH shows him/her an error message and the Student has to repeat the procedure
  - If the chosen team name is not available, the Student has to change it

Search for Project Teams

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing a list of projects
- *Event flow:*

1. the Student selects a project from the list
2. MPH shows the Student a list of project teams
3. the Student selects a project team from the list

- *Post-condition:* MPH shows the Student the information about the selected project team
- *Exceptions:*
  - If the are no projects teams corresponding to the selected project, MPH notifies the Student, who can select another project

Join Project Team

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing a list of project teams
- *Event flow:*

1. the Student selects a project team from the list
2. the Student executes the command to join the selected team
3. MPH shows a message asking the student to confirm his/her choice to join the project team
4. Student confirms his/her choice
5. MPH notifies the student that his/her request to join the selected team has been sent to the team members

- *Post-condition:* a membership request has been sent to all team members
- *Exceptions:*
  - If the desired project team does not accept any other members, the Student is notified by the system and he/she has the possibility to select another project team
  - If the teams presentation deadline has passed, MPH notifies the Student that it is not possible to join any project team

Accept/Decline New Membership Request

- *Participating actors:* Student
- *Pre-condition:* the Student is logged into the system and belongs to a project team
- *Event flow:*

1. the Student receives the notification of a new join request
2. MPH shows the Student a notification containing the information about a second student who wants to join the project team
3. the Student accepts/declines the new membership request
4. MPH shows a message asking the Student to confirm his/her choice
5. the Student confirms the choice
6. MPH notifies the Student that the desired choice has been executed

- *Post-condition:* the second Student is added to the project team if his/her request has been accepted, he is not added otherwise
- *Exceptions:*
  - If the teams presentation deadline has passed or the new membership request has been pending for too long, the request is automatically canceled

Allow/Block New Membership Requests

- *Participating actors:* Student
- *Pre-condition:* the Student is logged into the system and belongs to a project team
- *Event flow:*

1. the Student accesses the project team panel
2. MPH shows the Student the project team panel
3. the Student checks or unchecks the option to allow new membership requests
4. the Student saves the changes
5. MPH notifies the Student that the changes have been successfully saved

- *Post-condition:* the project team the Student belongs to will allow new membership requests only if the option is checked

- *Exceptions:*
  - If the teams presentation deadline has passed, MPH notifies the Student that no changes can be made to the project team

Leave Project Team

- *Participating actors:* Student
- *Pre-condition:* the Student is logged in and belongs to a project team
- *Event flow:*

1. the Student accesses the project team panel
2. MPH shows the Student the team panel
3. the Student executes the command to leave the project team
4. MPH shows the Student a message asking to confirm his/her choice
5. the Student confirms his/her choice
6. MPH notifies the student that he/she successfully left the project team

- *Post-condition:* the Student does not belong anymore to the project team
- *Exceptions:*
  - If the teams presentation deadline has passed, MPH notifies the Student that he/she can not leave the current project team

View Deliverables

- *Participating actors:* Student
- *Pre-condition:* the Student is logged into the system
- *Event flow:*

1. the Student executes the command to view the list of deliverables required by the current project
2. MPH shows the Student a page containing the deliverables required by the current project and the corresponding files delivered by the team he/she belongs to
3. the Student selects one or more files from the list
4. MPH lets the Student download the selected files

- *Post-condition:* the selected files are sent to the Student's client
- *Exceptions:*
  - If the project team the Student belongs to has not uploaded any files yet, the Student is notified by the system

Upload Artifact

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing the list of deliverables required by the project
- *Event flow:*

1. a Student executes the command to upload an artifact
2. MPH shows an upload form
3. the Student browses his/her computer
4. the Student selects the file to upload
5. MPH notifies the Student that the selected file has been uploaded correctly

- *Post-condition:* MPH permanently stores the uploaded file
- Exceptions:
  - If the current project is not active yet, MPH notifies him/her that it is necessary for the project to be active in order to upload a deliverable
  - If an input/output error occurs or the system loses its connection with the database, no file is transferred, Student is notified that the procedure encountered a problem and that he/she has to try another time

View Shared Information

- *Participating actors:* Student
- *Pre-condition:* the Student is viewing the list of deliverables required by the project
- *Event flow:*

1. the Student selects the option to view the artifacts delivered by another project team
2. MPH shows the Student at the bottom of the deliverables page a frame containing all the deliverables uploaded by the other teams, sorted by team
3. Student selects one or more files from the list
4. MPH lets the Student download the selected files

- *Post-condition:* the selected files are sent to the Student's client
- *Exceptions:*
  - If the other project team has not uploaded any files yet, the Student is notified by the system

Log Out

- *Participating actors:* Student
- *Pre-condition:* the Student is logged into the system
- *Event flow:*

1. the Student executes the command to log out from the system
2. MPH notifies the Student about the successful logout

- *Post-condition:* the Student is no more logged into the system
- *Exceptions:*
  - If an error occurs during the logout procedure MPH notifies the Student that he/she has to execute the command to log out another time

3.3.3 Professor

Log In

- *Participating actors:* Professor
- *Pre-condition:* the Professor, who is already registered into the system, accesses MPH
- *Event flow:*

1. the Professor executes the login command
2. MPH shows a login form
3. the Professor fills the form with username and password
4. MPH checks username and password and authenticates the Professor

- *Post-condition:* the Professor is logged into the system
- *Exceptions:*
  - If the inserted credentials are incorrect, MPH notifies the Professor that he/she has to repeat the procedure

Create Course

- *Participating actors:* Professor, System Administrator
- *Pre-condition:* the Professor wants to insert a new course held by him/her into the MPH system
- *Event flow:*

1. the Professor contacts the Admin asking him/her to create a new course
2. the Admin creates the new course

- *Post-condition:* a new course held by the Professor is inserted into the system
- *Exceptions:*
  - If the Admin is not able to create the new course, the Professor has to wait and try another time

Publish Project Description and Deadlines

- *Participating actors:* Professor
- *Pre-condition:*
- *Event flow:* the Professor is logged into the system

1. the Professor executes the command to publish a new project
2. MPH shows the Professor the list of courses held by him/her
3. the Professor selects a course from the list
4. MPH shows the Professor an input form
5. the Professor fills the form with project title, deliverables' deadlines' dates, the deadline penalty and a description
6. the Professor saves the changes
7. MPH notifies the Professor that the new project has been published correctly

- *Post-condition:* MPH inserts the new project details into the system
- *Exceptions:*
  - If the fields of the input form are filled incorrectly, the Professor has to repeat the operation

View Project Teams and Students

- *Participating actors:* Professor
- *Pre-condition:* the Professor is logged into the system
- *Event flow:*

1. the Professor executes the command to view the list of registered students and project teams
2. MPH shows the Professor a project search form
3. the Professor fills the form with the course and project name
4. MPH shows the Professor a list of projects matching the Professor's choice
5. the Professor selects a project from the list

- *Post-condition:* MPH shows the Professor a list of students and project teams participating in the selected project
- *Exceptions:*
  - If the key fields of the search form are not filled correctly, the Professor has to repeat the operation
  - If MPH does not find any entity matching the search form data, it notifies the Professor, who can perform a new search

Allow Information Sharing

- *Participating actors:* Professor
- *Pre-condition:* the Professor is viewing a list of project teams
- *Event flow:*

1. the Professor selects the project team who should have visibility on the artifacts produced by another team
2. MPH shows the Professor a list of teams belonging to the same project of the team previously selected
3. the Professor selects the teams whose files will be accessible by the team selected at point 1
4. MPH notifies the Professor that the operation has been successful

- *Post-condition:* the first team can access the artifacts produced by the second teams
- *Exceptions:*
  - No exceptions are expected

View Deliverables

- *Participating actors:* Professor
- *Pre-condition:* the Professor wants to download all the files delivered by a team and he/she is logged into the system
- *Event flow:*

1. the Professor executes the command to view the deliverables corresponding to a project
2. MPH shows the Professor the list of courses held by him/her
3. the Professor selects a course from the list

- *Post-condition:* MPH shows the Professor the list of project deliverables
- *Exceptions:*
  - If the selected project does not contain any delivered files yet, the Professor is notified by the System

View Deliverables per team

- *Participating actors:* Professor
- *Pre-condition:* the Professor is viewing the list of deliverables corresponding to a project
- *Event flow:*

1. the Professor executes the command to view all the files delivered by a team
2. MPH shows the Professor a search form
3. the Professor fills the form with the name or the number of the desired team
4. MPH shows the Professor a list of project teams matching the Professor's choice
5. the Professor selects a project team from the list
6. MPH shows the Professor the list of files delivered by the selected project team
7. the Professor selects one or more files to download

- *Post-condition:* the selected files are sent to the Professor's client

- *Exceptions:*
  - If the key fields of the search form are not filled correctly, the Professor has to repeat the operation
  - If MPH does not find any entity matching the search form data, it notifies the Professor, who can perform a new search
  - If the selected project team has not delivered any files yet, the Professor is notified by the System

View Deliverables per Type

- *Participating actors:* Professor
- *Pre-condition:* the Professor is viewing the list of deliverables corresponding to a project
- *Event flow:*

1. the Professor selects a deliverable type
2. MPH shows the Professor a list of artifacts of the desired type, uploaded by all project teams
3. the Professor selects one or more files to download

- *Post-condition:* the selected files are sent to the Professor's client
- *Exceptions:*
  - If no artifacts of the desired type have been uploaded, the Professor is notified by the system

Evaluate Artifact

- *Participating actors:* Professor
- *Pre-condition:* the Professor is viewing a list of delivered files
- *Event flow:*

1. the Professor selects an artifact to evaluate
2. MPH shows the Professor an evaluation form
3. the Professor fills the form with a score and optional notes
4. the Professor saves the changes
5. MPH notifies the Professor that changes have been saved correctly

- *Post-condition:* the evaluation data are saved permanently into the system
- *Exceptions:*
  - If the fields of the evaluation form are not filled correctly, the Professor has to repeat the operation

Log Out

- *Participating actors:* Professor
- *Pre-condition:* the Professor is logged into the system
- *Event flow:*

1. the Professor executes the command to log out from the system
2. MPH notifies the Professor about the successful logout

- *Post-condition:* the Professor is no more logged into the system
- *Exceptions:*

  If an error occurs during the logout procedure MPH notifies the Professor that he/she has to execute the command to log out another time

3.4 Functional Requirements

From the use case analysis, the main functions of the MPH software system involves three classes of users: students, professors and system administrators.

3.4.1 Professors-Related Functions

After the system administrator registers a professor into the system, the system allows the professor to perform the following set of actions:

- login into the system
- publish a project description defining the set of corresponding deliverables characterized by a name and a specific deadline
- see the list of registered project teams and the students composing them
- download the delivered files from two views, the first one organizes the deliverables per team, the second one per deliverable type
- evaluate the deliverables of each team with a score from 1 to 10
- enable some simple information sharing between teams, allowing a team to have visibility on the deliverables produced by another team

Moreover the system can compute the final score obtained by each team as an average of the intermediate scores and can automatically assign score 1 to artifacts not delivered or apply a penalty on the score of late deliverables proportional to the delay time.

3.4.2 Students-Related Functions

MPH provides students with the following functions:

- autonomous registration to the system
- access to the profile information
- join a project team
- access the information and functionalities associated to the team the student belongs to
- submit a deliverable (i.e. a single file) by uploading it into the system
- access the shared deliverables produced by another team that he/she is allowed to view

3.4.3 System-Administrators Functions

MPH provides sysadmins with the following functions:

- Possibility to have complete read/write access to the database of information about the MPH system
- See the status of every transaction made by a user
- See how many users are connected to the MPH system
- Start and stop the MPH system

3.5 Non-Functional Requirements

3.5.1 Reliability

MPH needs to ensure the persistence of the data inserted by the users, saving the MPH system status during the period of time while the system is offline.

3.5.2 Security

There system must provide an authentication system capable of hiding all the operations not allowed to a particular user.

3.5.3 Performance

Access times to the system should be kept reasonably low, and the system must ensure sufficient reactivity to the user input, in the order of tens of seconds.

3.5.4 Usability

The ease and the human interaction with the machine assumes a role of fundamental importance for the success of the final application. The use of the system should be intuitive, and for each type of user each available function must be implemented to provide a painless and immediate access.

4 Model

4.1 Static Models

4.1.1 Class Diagram

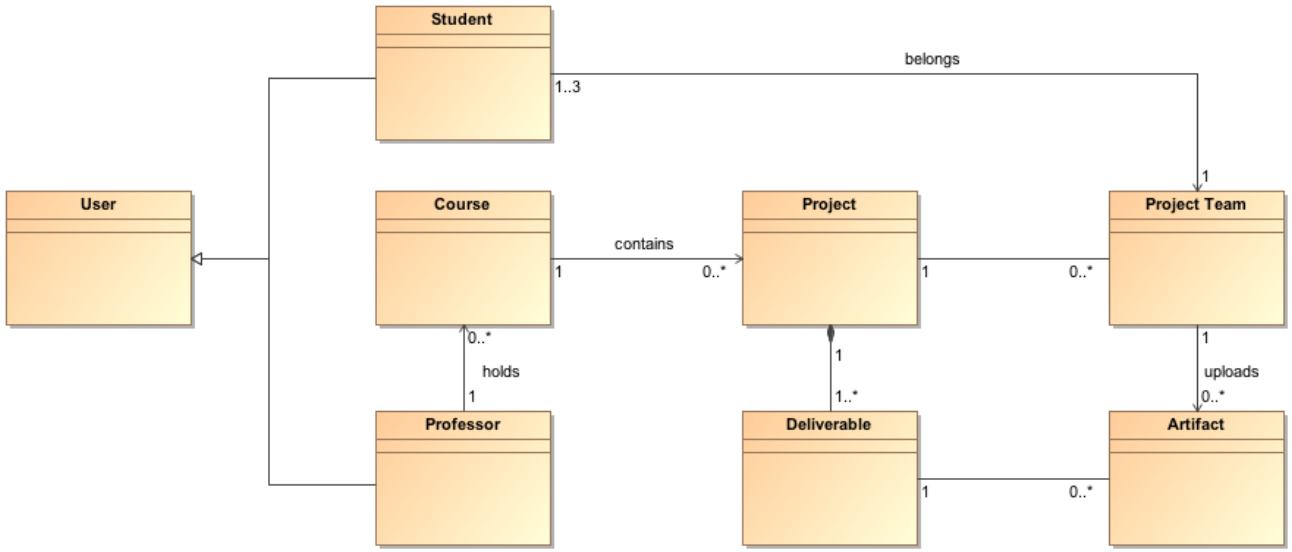

Figure 2: MPH Class Diagram

4.2 Dynamic Models

4.2.1 Sequence diagrams

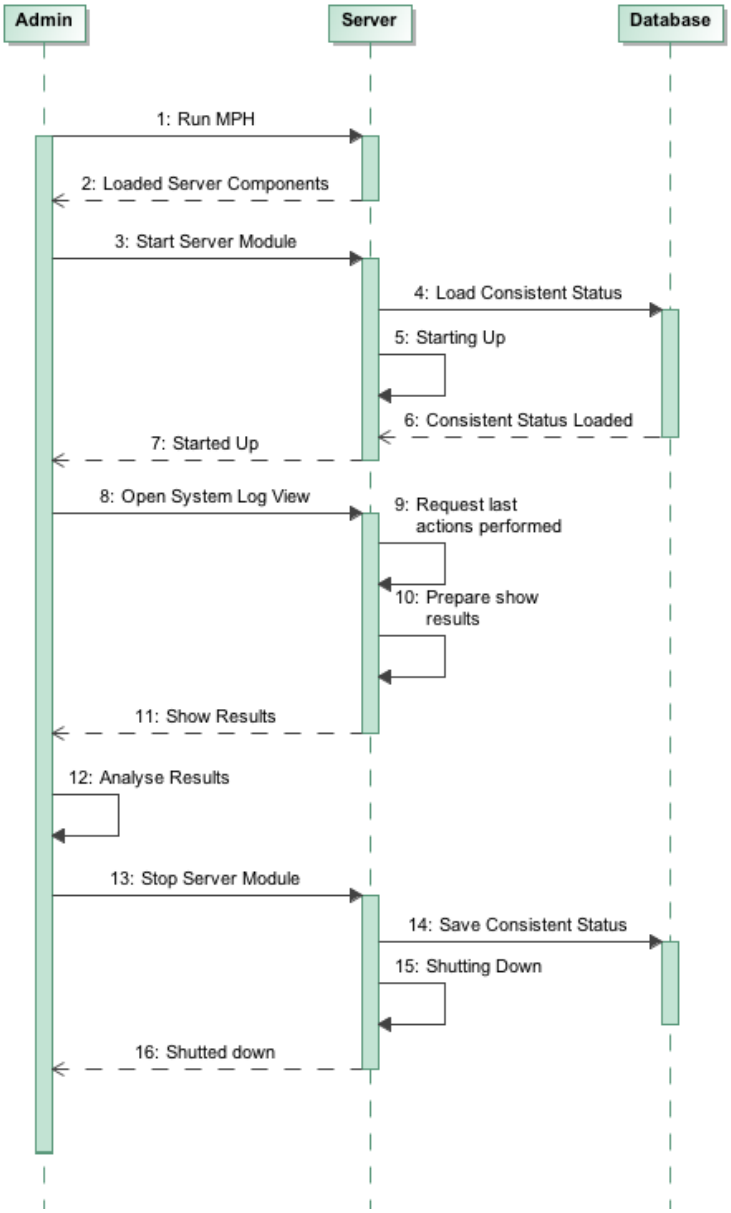

Figure 3: The System Administrator starts the MPH server module and checks the System Log

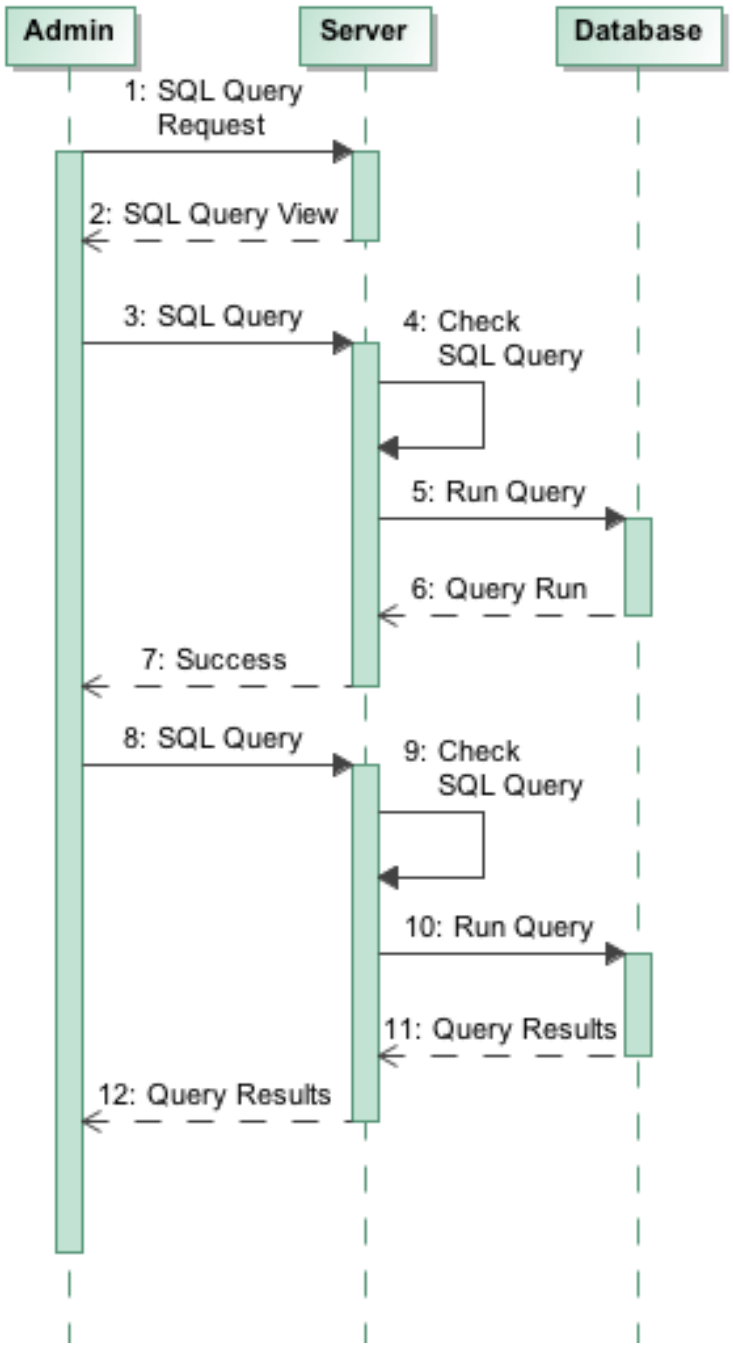

Figure 4: The System Administrator checks a Database Table

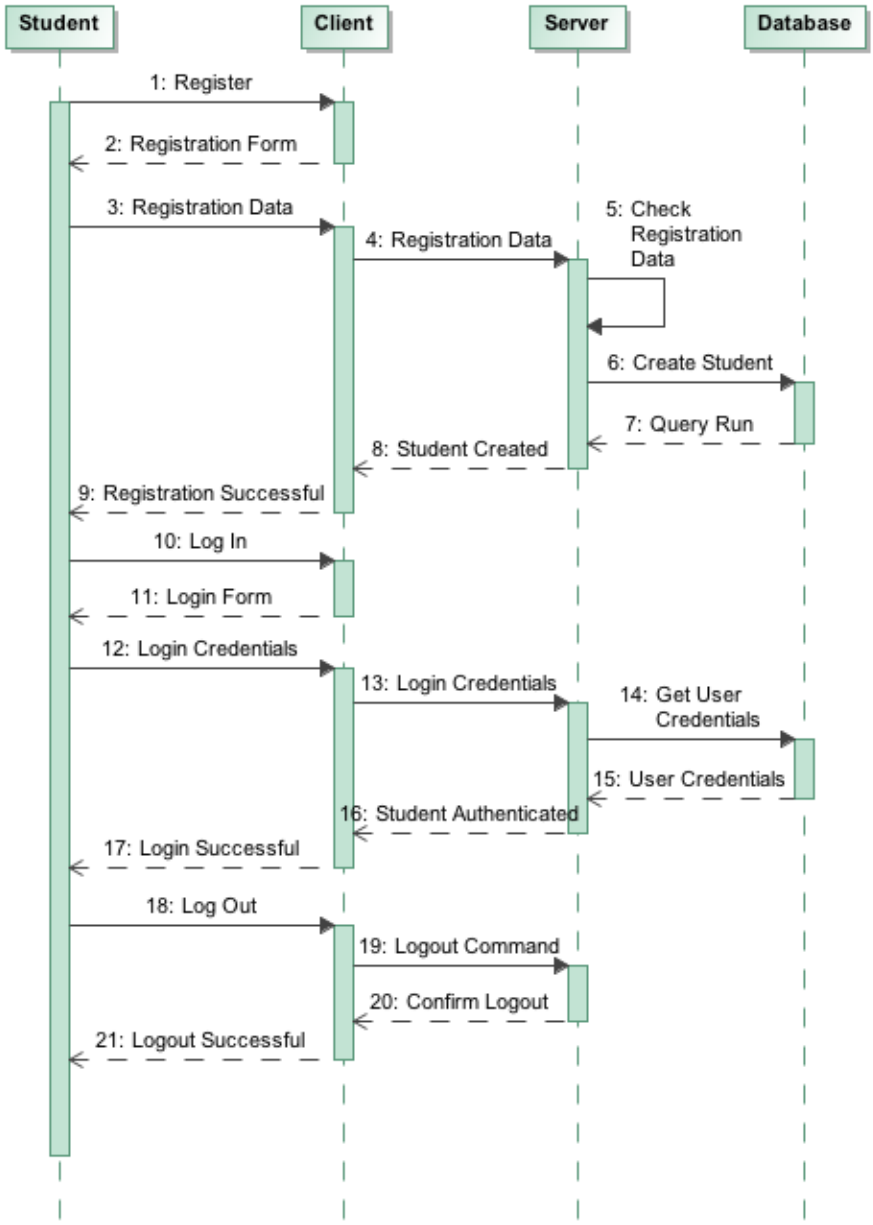

Figure 5: Registration, Login and Logout of a Student

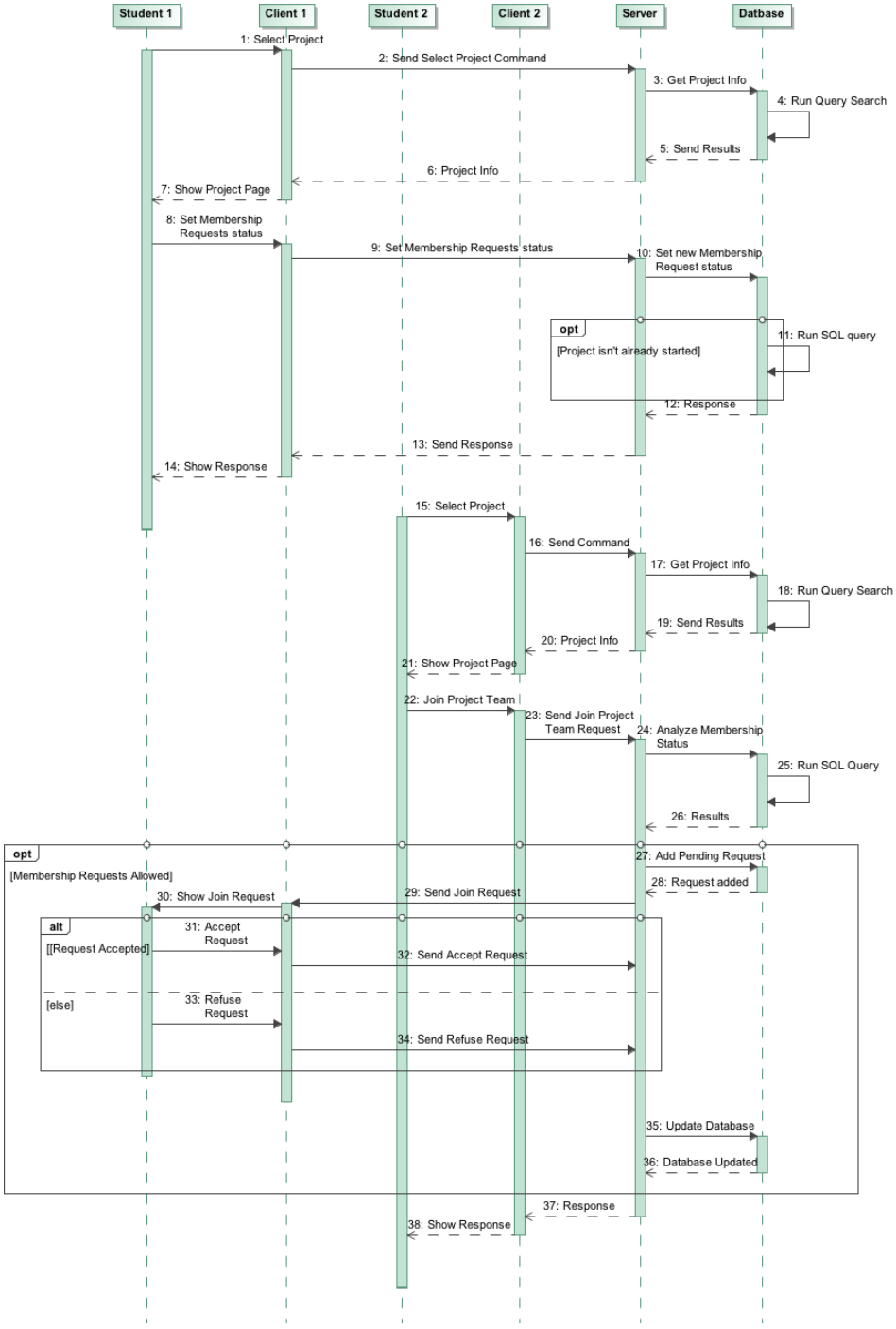

Figure 6: Team Membership Request Process

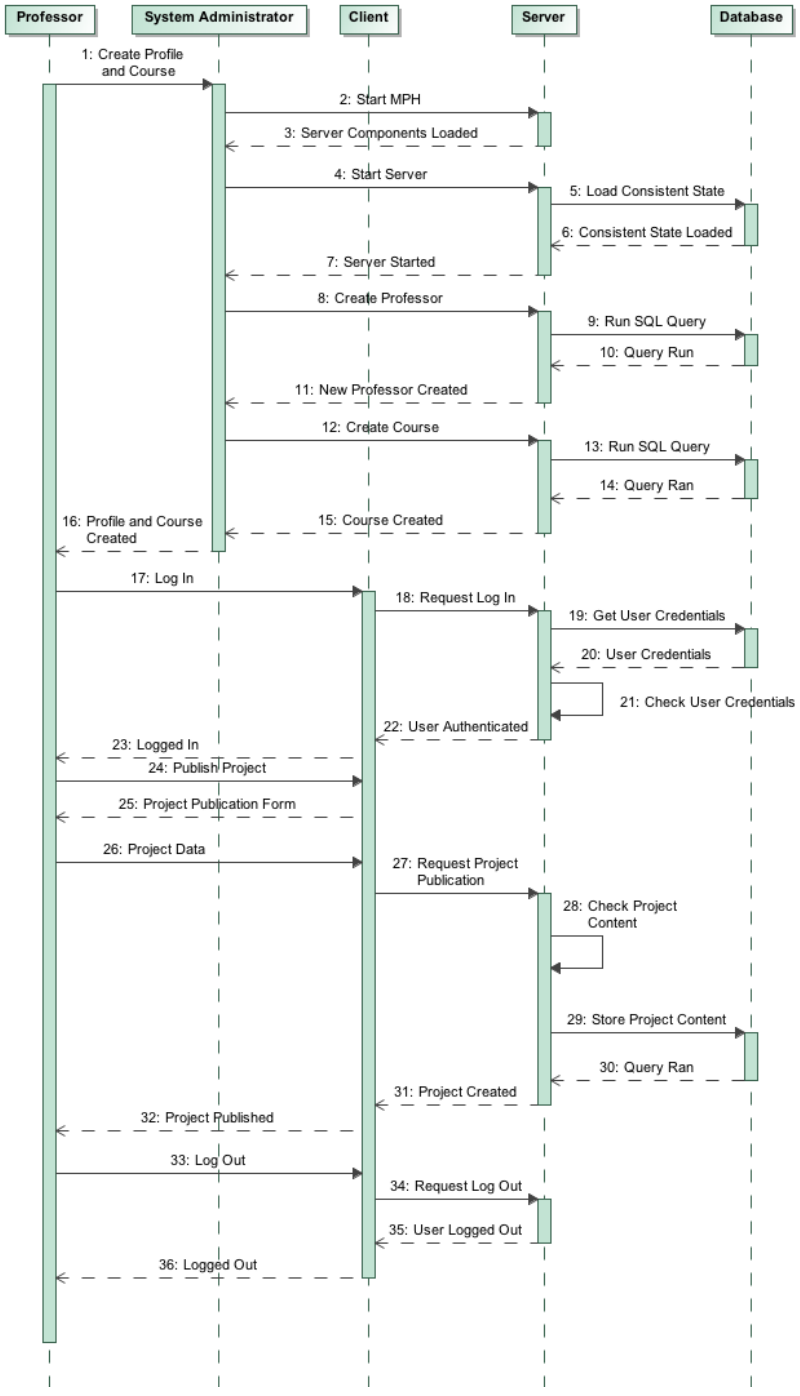

Figure 7: Registration of a new professor and definition of a new project

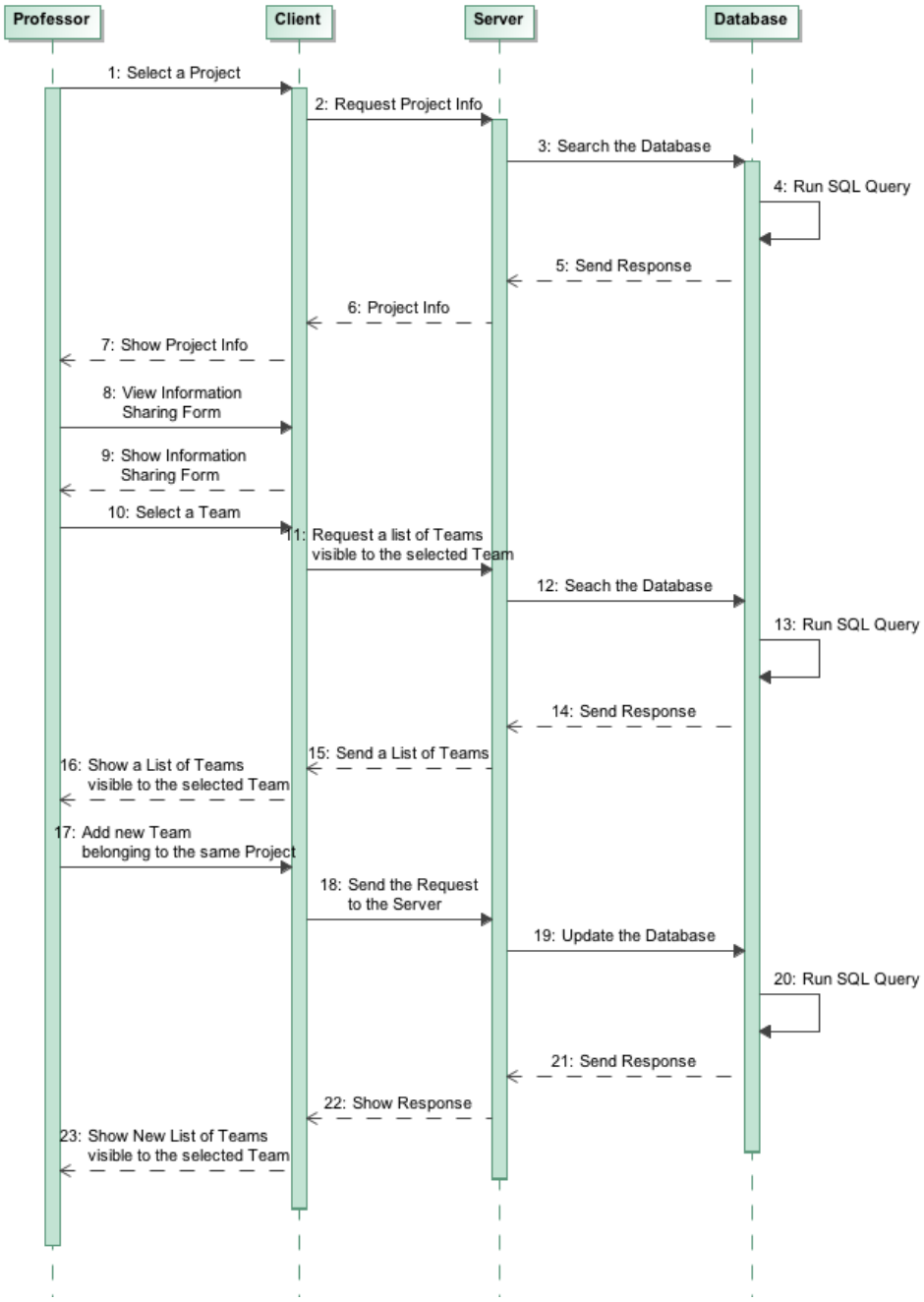

Figure 8: A Professor sets Information Sharing between Teams

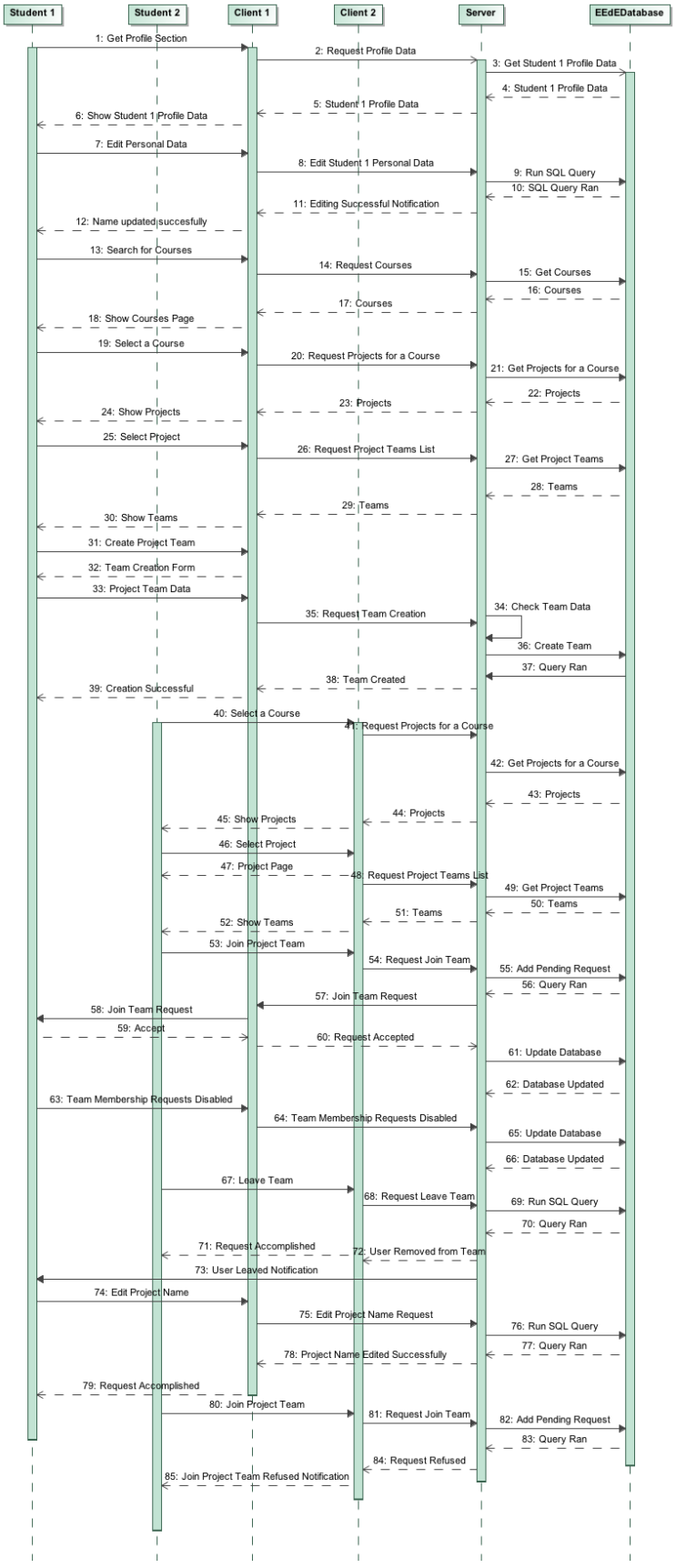

Figure 9: A student creates a team and another student joins it

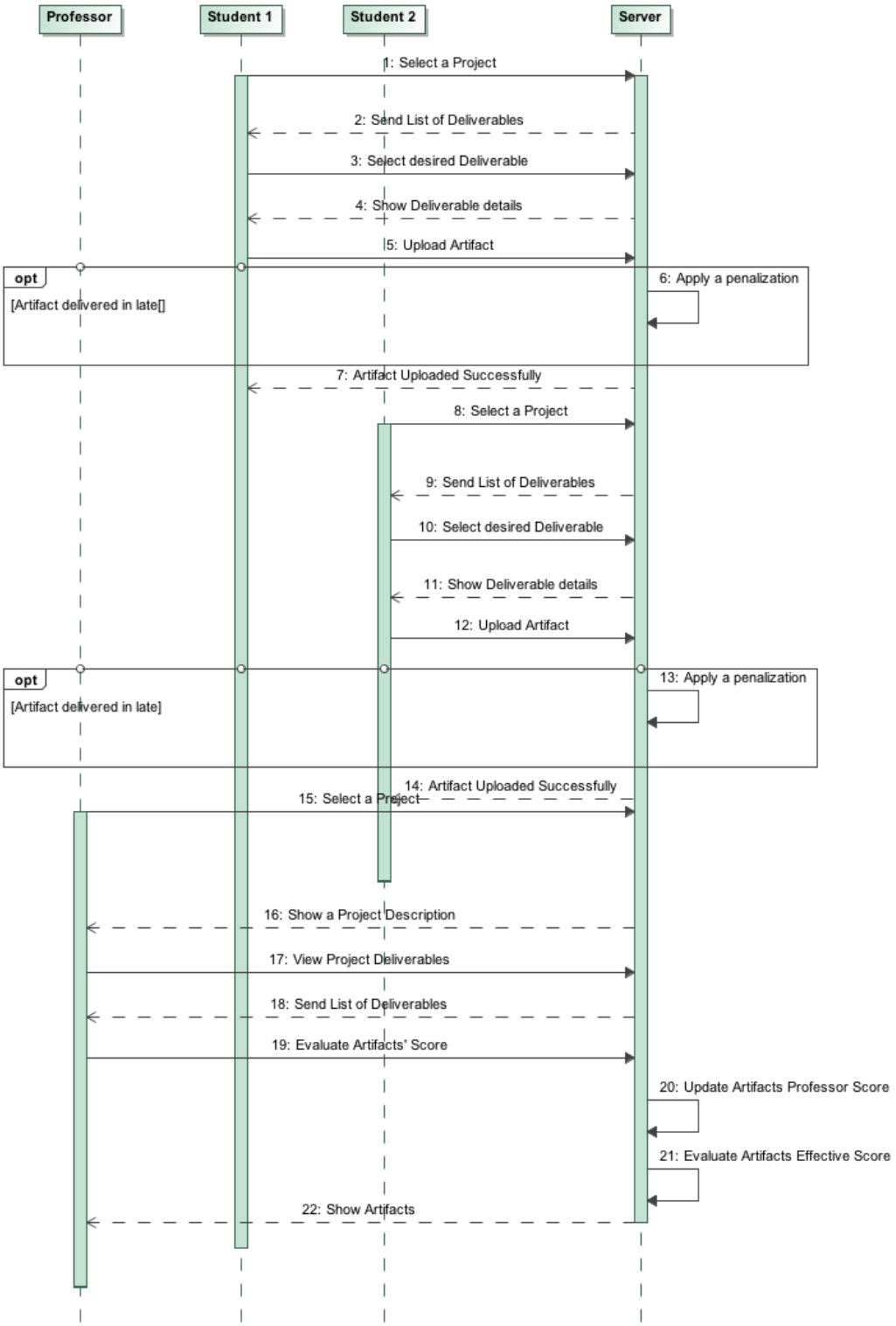

Figure 10: Complex example of interaction between professors and teams

4.2.2 Activity diagrams

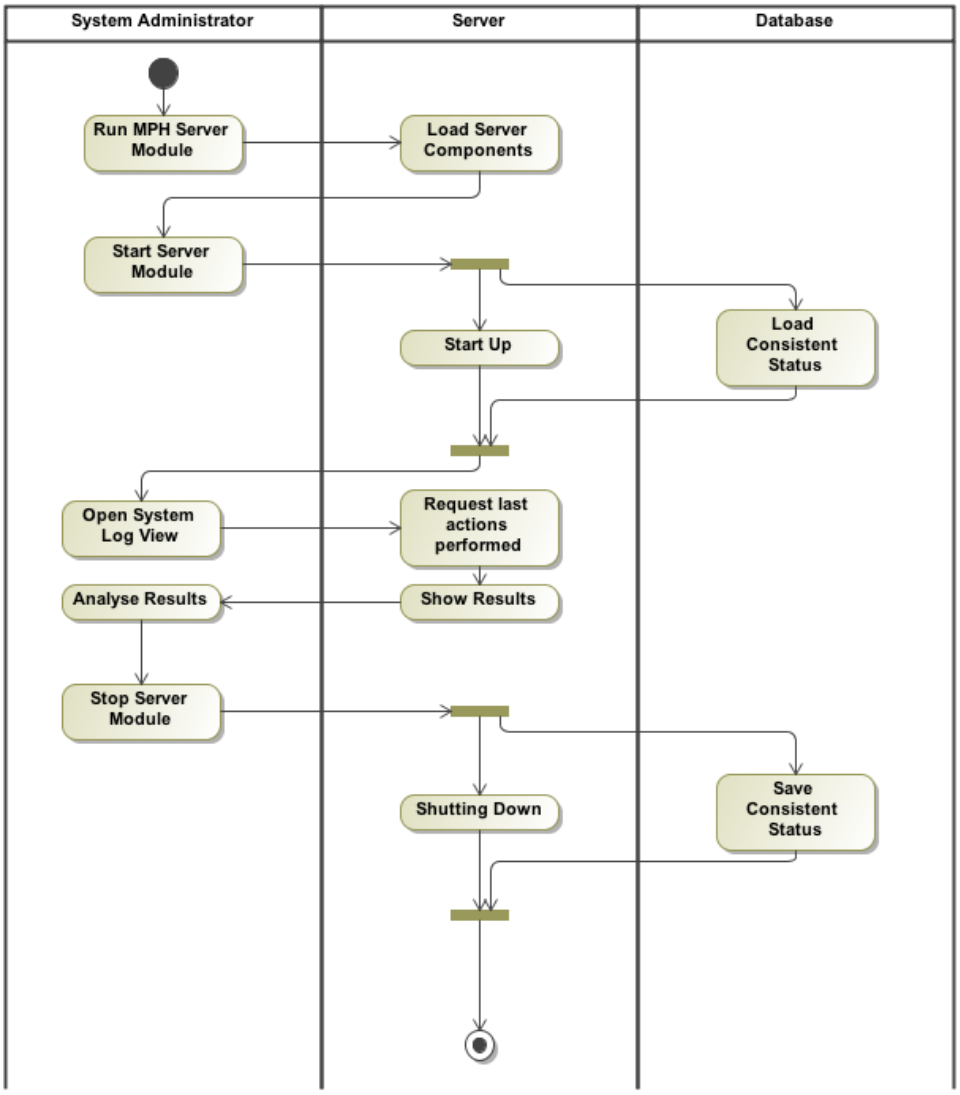

Figure 11: The System Administrator starts the MPH server module and checks the System Log

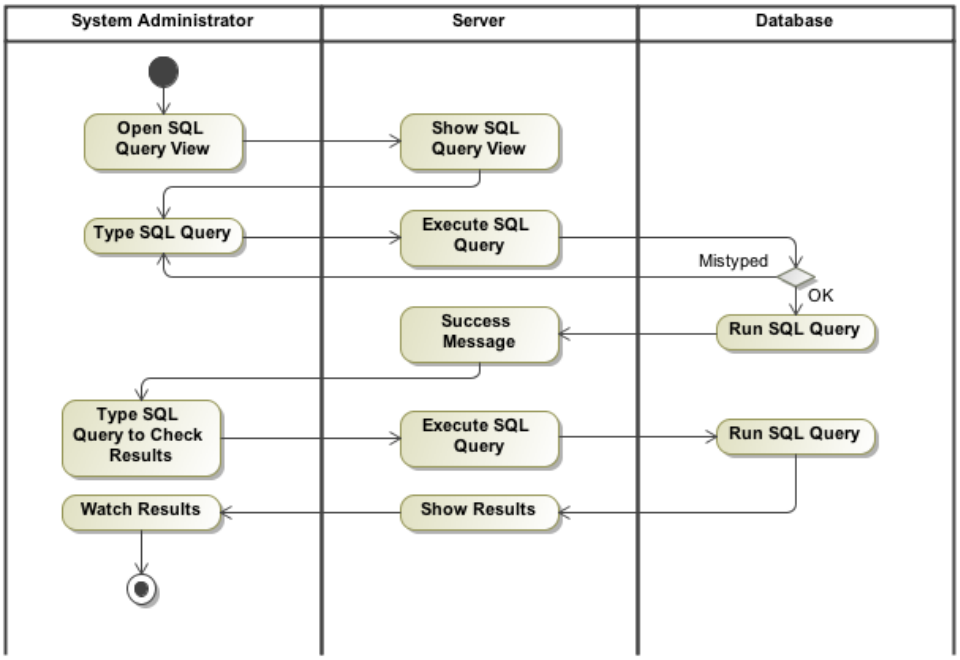

Figure 12: The System Administrator checks a Database Table

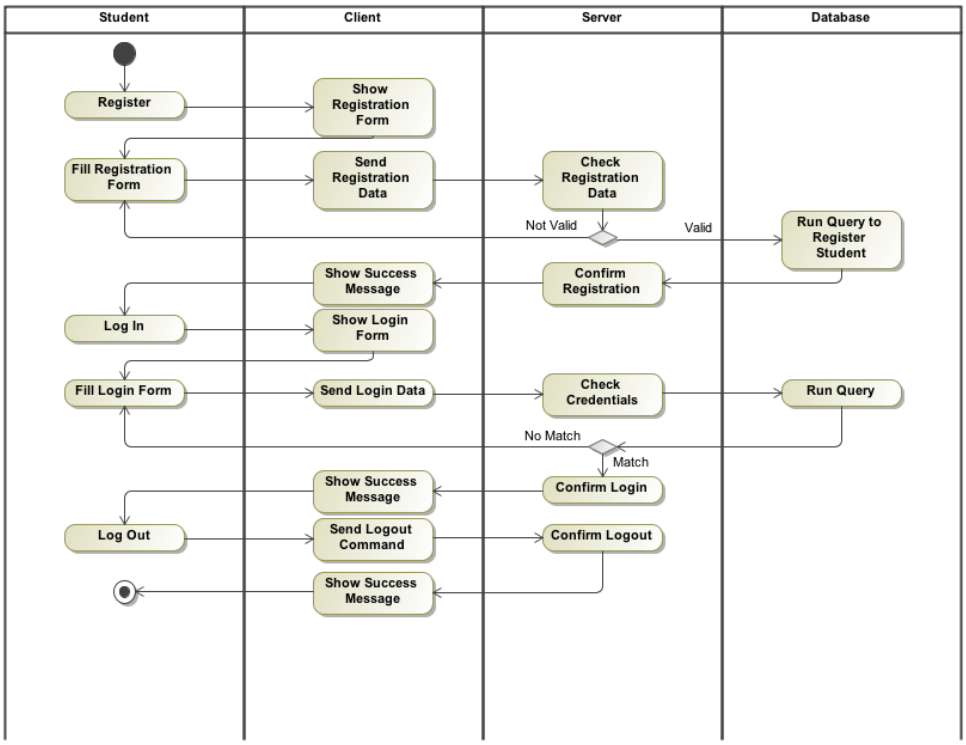

Figure 13: Registration, Login and Logout of a Student

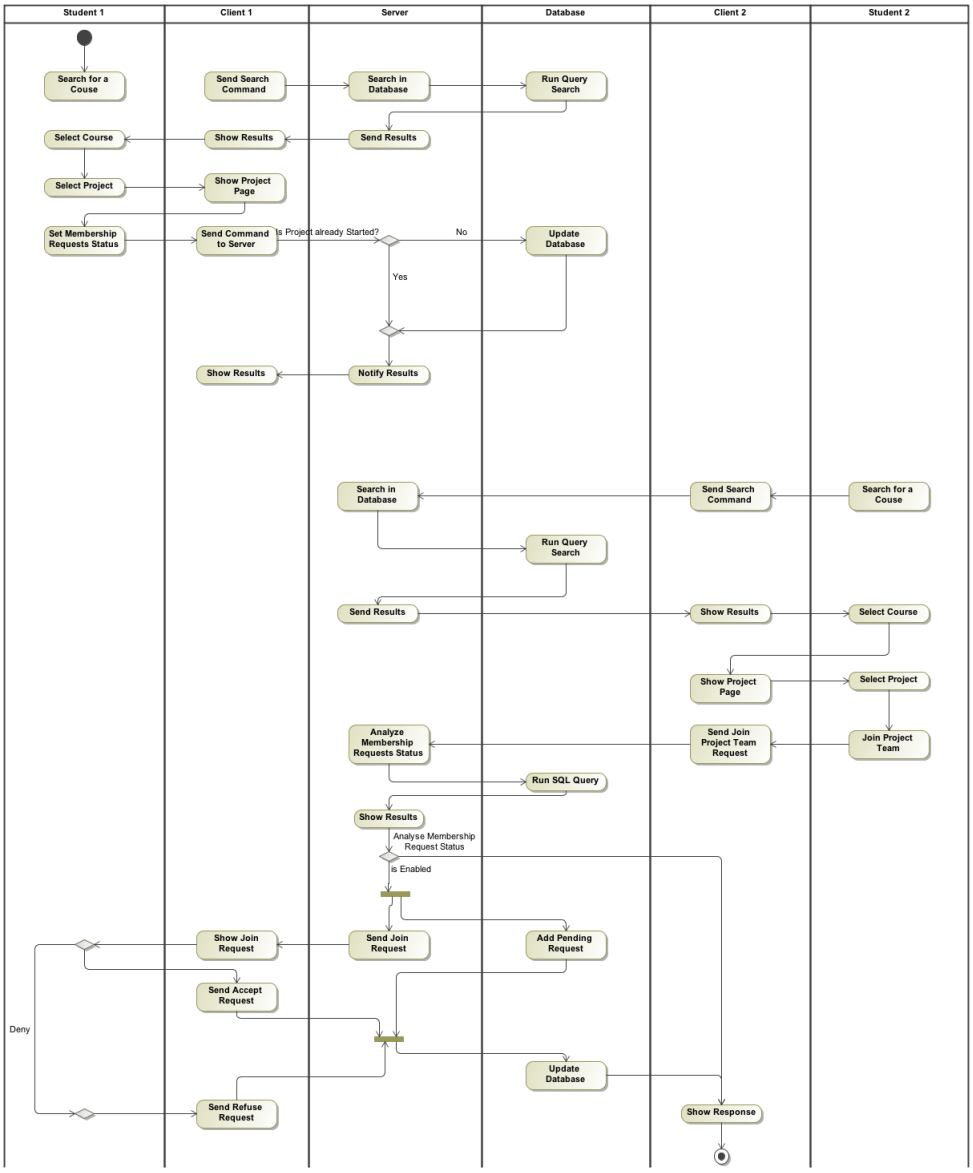

Figure 14: Team Membership Request Process

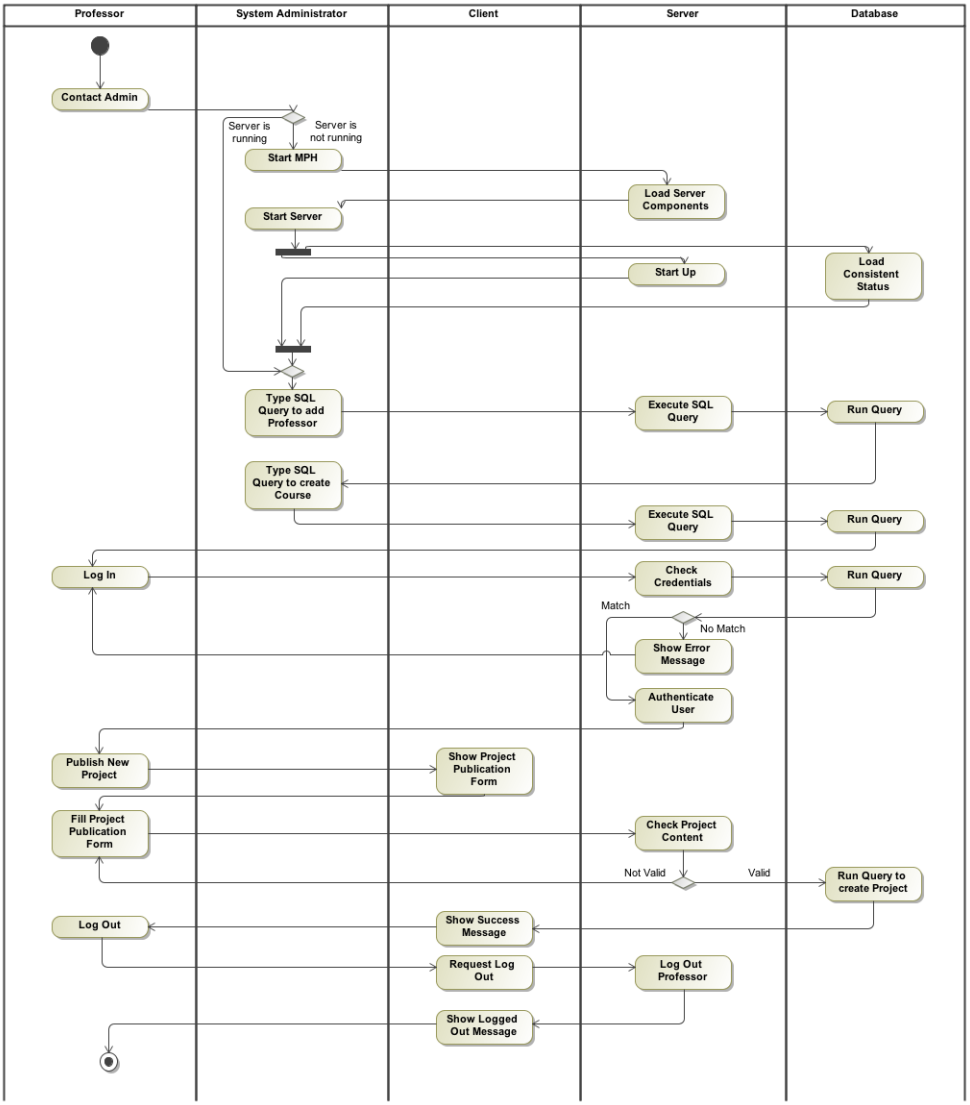

Figure 15: Registration of a new professor and definition of a new project

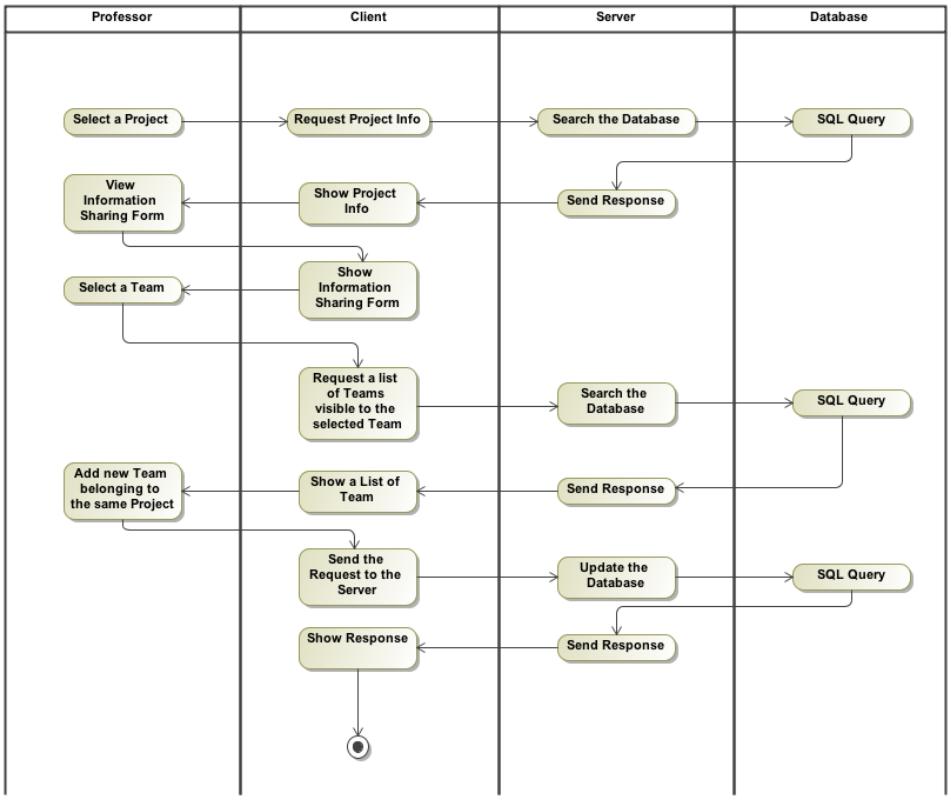

Figure 16: A Professor sets Information Sharing between Teams

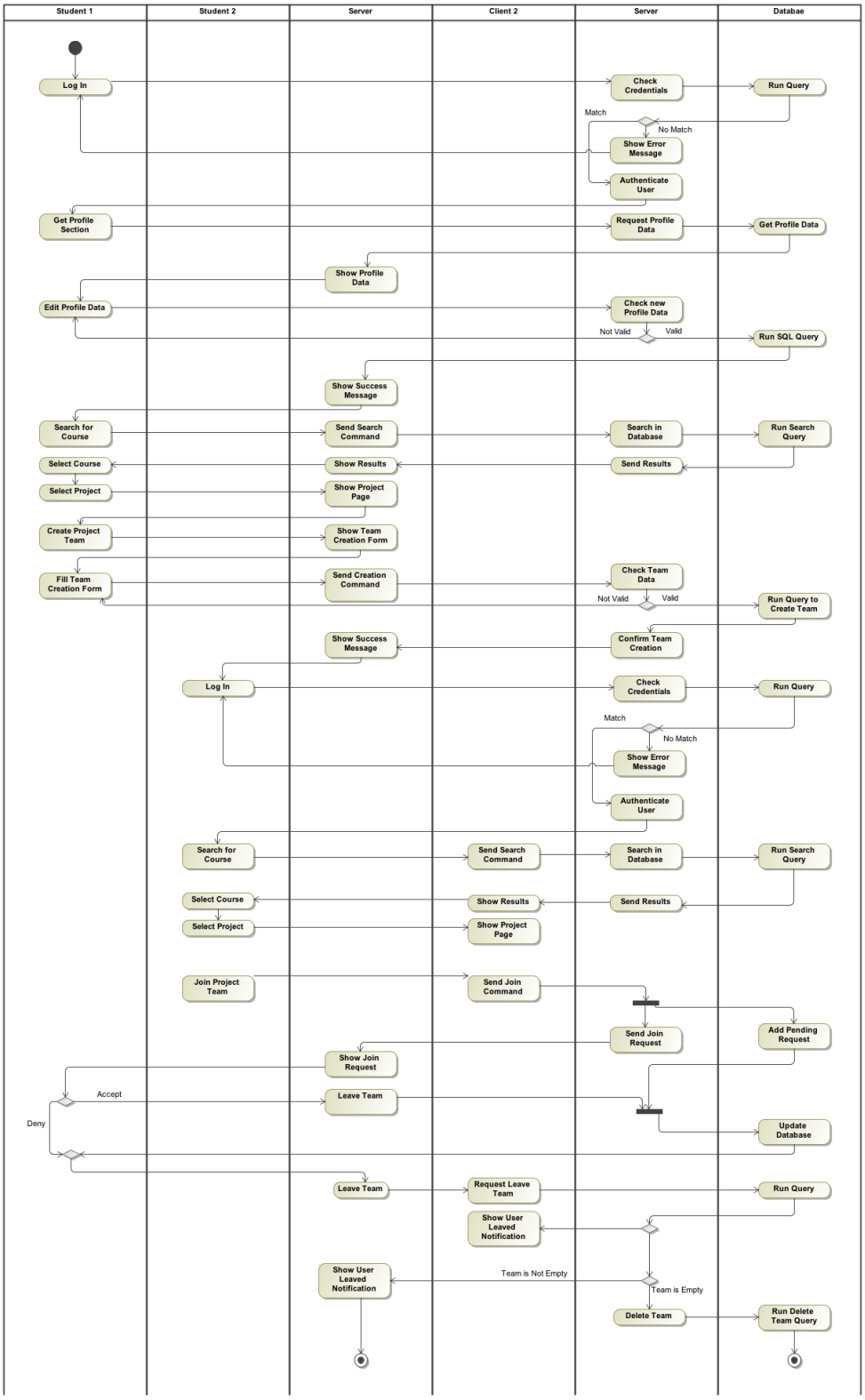

Figure 17: A student creates a team and another student joins it

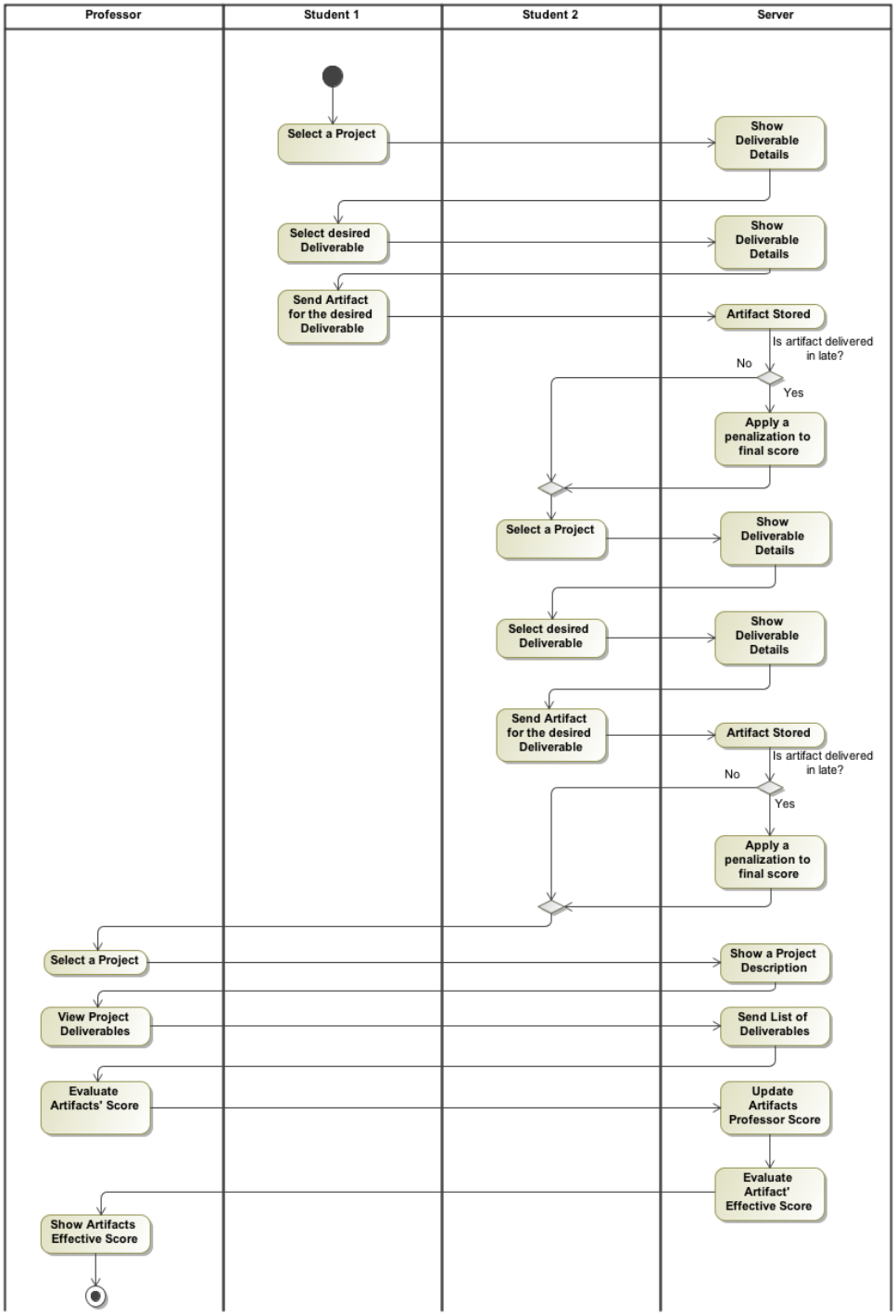

Figure 18: Complex example of interaction between professors and teams

4.2.3 State Charts

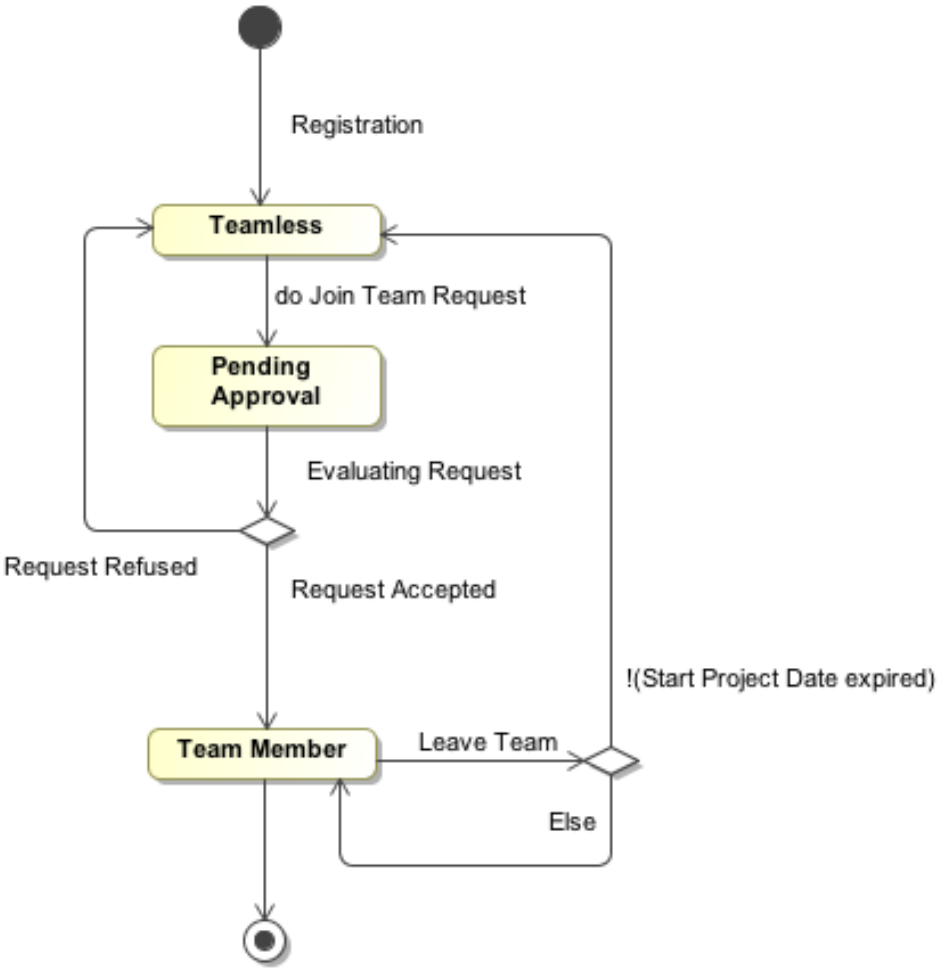

Figure 19: Student State Chart

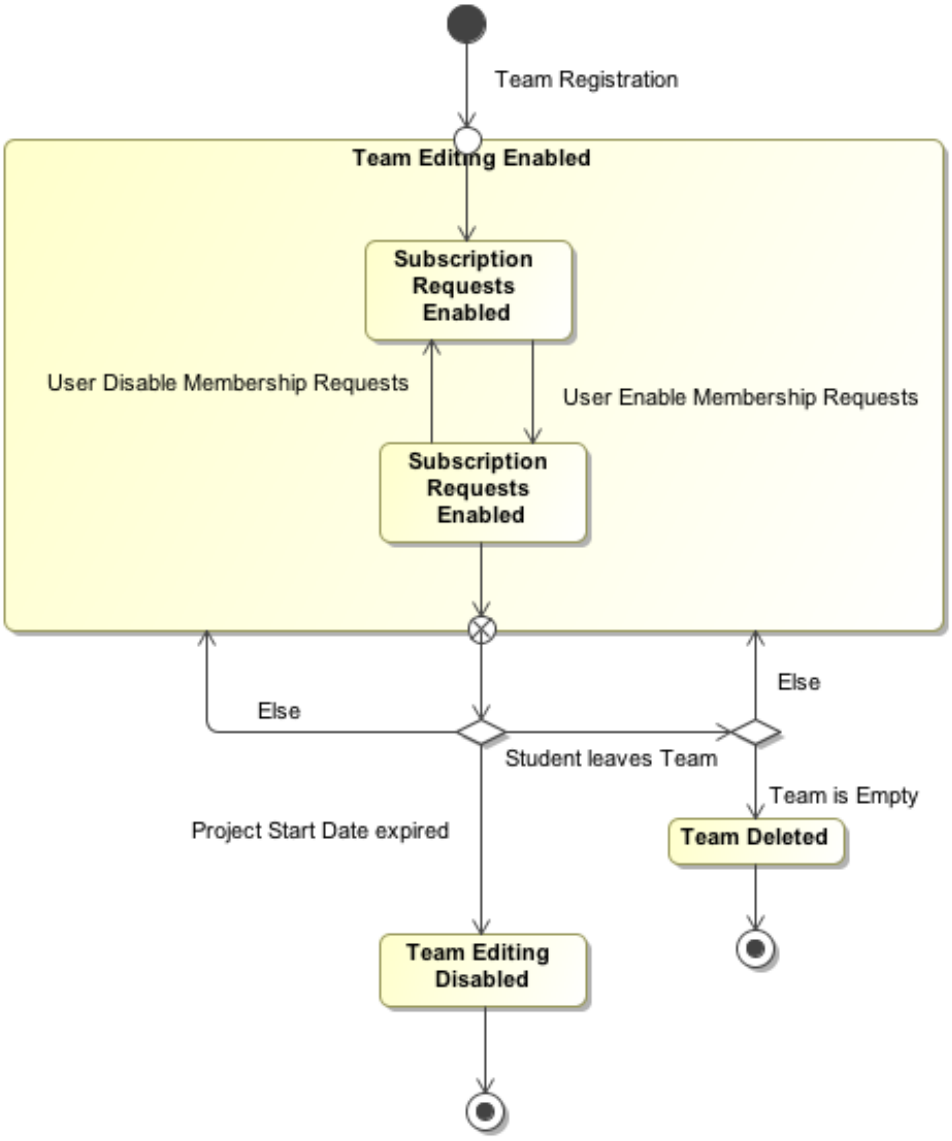

Figure 20: Team State Chart

5 Specifications

5.1 Design Constraints

5.1.1 Users

- Every user must have a unique username and a password
- Every user can have a First Name, a Last Name, a Phone Number and an E-Mail

5.1.2 Professor

- Every professor can be associated to a set of courses

5.1.3 Students

- Every student can not be part of more than one team simultaneously per project: that means that if a student is in a team, he/she can not join or create other teams for the same project
- Every student member of a team can leave a team only before the ”Project Start Date”

5.1.4 Teams

- Every team is composed by one to three students
- Every team has a unique name per project
- At its creation, a team is set to accept ”Membership Requests”
- At any time before the ”Project Start Date”, any member of the team can set the team to accept ”Membership Requests”or not
- When the ”Project Start Date” is reached, the team is automatically set to not accept ”Membership Requests”, the internal composition and the name of the team can not be changed anymore
- When a team can not accept ”Membership Requests” no other student can join the team
- Every team has a number of artifacts minor or equal to the number of deliverables of the project
- If in a team, after its creation, there are no more students, the team is automatically deleted
- Every team can only share information with teams related to the same project

5.1.5 Courses

- Every course must be held only by one Professor
- Every course has a set of students associated to it
- Every course can have projects

5.1.6 Projects

- Every project must be defined only by one professor
- Every project must be associated to only one Course
- Every project must have a ”Creation Date”, a ”Start Date” and a ”End Date”: the ”End Date” must be posterior the ”Start Date” and both must be posterior to the project's creation date (recorded by the server module).
- Every project must have some Deliverables
- Every project can have a set of teams associated to it

5.1.7 Deliverables

- Every deliverable must have only one deadline associated to it, that is a tern made up by of day, month and year.
- Every deliverable must be associated to only one project
- Every deadline must follow the Project Start Date and precede the Project End Date

5.1.8 Artifacts

- Every artifact must be assigned to only one deliverable
- Every artifact must belong to only one team
- Every artifact can have a score from 1 to 10
- Every late delivered artifact has a got a penalty proportional to the days of delay, but the total score must always be equal or greater than one

5.2 Alloy

5.2.1 Code

```
module MPH_world

//Support signatures

//Date
sig Date {}

//Personal Data
sig PersonalData {}

//ID
sig ID {}

//Name
sig Name{}

//Score
sig Score {}

//Signatures

//User
abstract sig User {
        id: one ID,
        personalData: lone PersonalData,
}

//Professors
sig Professor extends User {
        courses: set Course,
                        //every professor can be
           associated to a set of courses
}

//Courses
sig Course{
        projects: set Project,
                        //every course can have
           projects
}

//Projects
```

```
39 sig Project {
40         deliverables: some Deliverable, //every
               project must have some Deliverables
41         teams: set Team,
                                                 //every
               project has a set of teams associated to it
42 }
43
44 //Students
45 sig Student extends User{}
46
47 //Team
48 sig Team {
49         name: one Name,
50         students: set Student,
51         artifacts: set Artifact,
                                        //every team has
               uploaded a set of artifacts
52         shared: set Team,
                                        //every team can
               share information with another one
53 } {#students >=1 and #students <= 3}
54
55 //Deliverables
56 sig Deliverable {
57         deadline: one Date,
                                        //every derivable
               must have only one deadline associated to
               it
58 }
59
60 //Artifacts
61 sig Artifact {
62         deliverable: one Deliverable,
63         score: lone Score,
64 }
65
66 //Facts
67
68 //ID's of professors and students must be unique
69 fact uniqueID {
70         all p: Professor, s: Student | p.id != s.id
71         all s1, s2: Student | s1 != s2 implies s1.id
               != s2.id
72         all p1, p2: Professor | p1 != p2 implies p1.id
               != p2.id
```

```
73 }
74
75 //for every course there is a professor associated to
      it
76 fact noCourseWithoutProfessor {
77         all c: Course | one p: Professor | c in p.
              courses
78 }
79
80 //for every project there is a course associated to it
81 fact noProjectWithoutCourse {
82         all p: Project | one c: Course| p in c.
              projects
83 }
84
85 //for every deliverable there is a project associated
      to it
86 fact noDeliverableWithoutProject {
87         all d: Deliverable | one p: Project| d in p.
              deliverables
88 }
89
90 //every student can not belong to teams related to the
       same project
91 fact uniqueProjectTeamPerStudent {
92         all s: Student, t1, t2: Team | (s in t1.
              students and s in t2.students and t1 != t2)
               implies no p: Project | t1 in p.teams and
              t2 in p.teams
93 }
94
95 //every team must be associated to only one project
96 fact uniqueProjectPerTeam {
97         all t: Team | no p1, p2: Project | p1 != p2
              and t in p1.teams and t in p2.teams
98         all t: Team | one p: Project | t in p.teams
99 }
100
101 //every project must be associated to only one course
102 fact uniqueCoursePerProject {
103         all p: Project | no c1, c2: Course | c1 != c2
              and p in c1.projects and p in c2.projects
104 }
105
106 //a team can not be composed by students with the same
       ID
```

```
107 fact uniqueStudentsPerTeam {
108         all t: Team, s1,s2: Student |  (s1 in t.
                students and s2 in t.students and s1 != s2)
                 implies s1.id != s2.id
109 }
110
111 //every team has a unique name per project
112 fact uniqueTeamNamePerProject {
113         all p: Project, t1,t2: Team | (t1 in p.teams
                and t2 in p.teams and t1 != t2) implies t1.
                name != t2.name
114 }
115
116 //every artifact must belong to only one team
117 fact uniqueTeamPerArtifact {
118         all a: Artifact | one t: Team | a in t.
                artifacts
119         all a: Artifact | no t1, t2: Team | t1 != t2
                and a in t1.artifacts and a in t2.artifacts
120 }
121
122 //every artifact delivered by a team must be
       associated to different deliverables belonging to
       the same project
123 fact uniqueDeliverablePerProjectTeamArtifact {
124         all a1, a2 : Artifact, t: Team | (a1 != a2 and
                 a1 in t.artifacts and a2 in t.artifacts)
                implies a1.deliverable != a2.deliverable
125         all a: Artifact, t: Team, p: Project | t in p.
                teams implies a.deliverable in p.
                deliverables
126 }
127
128 //every deliverable must be associated to only one
       project
129 fact uniqueProjectPerDeliverable {
130         no d: Deliverable, p1, p2: Project | p1 != p2
                and d in p1.deliverables and d in p2.
                deliverables
131 }
132
133 //every team has a number of artifacts minor or equal
       to the number of deliverables of the project
134 fact maxArtifactsPerTeam {
135         all t: Team, p: Project | t in p.teams implies
                 #t.artifacts <= #p.deliverables
```

```
136 }
137
138 //every team can not share information with itself
139 fact noSelfSharing{
140         all t: Team | no t1: Team | t1  in t.shared
               and t1 = t
141 }
142
143 //every team can only share information with teams
       related to the same project
144 fact noSharingBetweenDifProjects{
145         all t1, t2: Team, p: Project | (t1 in p.teams
               and (t1 in t2.shared or t2 in t1.shared))
               implies t2 in p.teams
146 }
147
148 //no lone signatures
149 fact noLoneSignatures {
150
151         all d: Date | one del: Deliverable | del.
               deadline = d                    //Date vs
               Deliverable
152         all n: Name | one t: Team | t.name = n

                                //Name vs Team
153         all s: Score | one a: Artifact | a.score = s

                                //Score vs Artifact
154         all pd: PersonalData | one u: User | u.
               personalData = pd         //Personal Data vs
                User
155         all i: ID | one u: User| u.id = i

                                                          //
               ID vs User
156
157 }
158
159 //Assertions
160
161 //every team contains at least one student but no more
       than three
162 assert studentsInTeam {
163         no t: Team | #t.students < 1 or #t.students >
               3
164 }
```

```
165
166 check studentsInTeam
167
168 //if there are no students, there are no teams
169 assert  noStudentsNoTeams{
170         #Student = 0 implies #Team = 0
171 }
172
173 check noStudentsNoTeams
174
175 //there are no project teams associated to zero or
       more than two projects
176 assert projectsPerTeam {
177         all t: Team | one p: Project | t in p.teams
178         all t: Team | no p1, p2: Project | p1 != p2
              and t in p1.teams and t in p2.teams
179 }
180
181 check projectsPerTeam
182
183 //there are no artifacts associated to deliverables
       belonging to a project different from the project
       associated to the team
184 assert artifactsProject {
185         all a: Artifact, t: Team, p: Project | (a in t
              .artifacts and t in p.teams) implies a.
              deliverable in p.deliverables
186 }
187
188 check artifactsProject
189
190 //if there are no projects, there are no artifacts
191 assert  noProjectsNoArtifacts {
192         #Project = 0 implies #Artifact = 0
193 }
194
195 check noProjectsNoArtifacts
196
197 //a team can not share information with teams
       belonging to a different project
198 assert noDifSharing {
199         all t1, t2 : Team | no p1,p2: Project | (t1 in
               t2.shared or t2 in t1.shared) and p1 != p2
               and t1 in p1.teams and t2 in p2.teams
200 }
201
```

```
202 check noDifSharing
203
204 //Predicates
205
206 pred showWorld1(){
207
208         #Deliverable > 5
209         #Professor > 1
210         #Team > 2
211         #Course > 1
212         #Artifact > 5
213 }
214
215
216 pred showWorld2(){
217
218         all c: Course | #c.projects > 1
219
220         #Project > 3
221         #Professor > 1
222         #Team > 2
223         #Course > 2
224         #Student > 4
225
226 }
227
228 pred showWorld3(){
229
230         all c: Course | #c.projects = 1
231         all c: Project | #c.teams > 0 and #c.
               deliverables> 0
232
233
234         #Project = 3
235         #Professor = 1
236         #Course > 2
237         #Student > 4
238
239 }
240 run showWorld1 for 7
241 run showWorld2 for 7
242 run showWorld3 for 7
```

5.2.2 Worlds

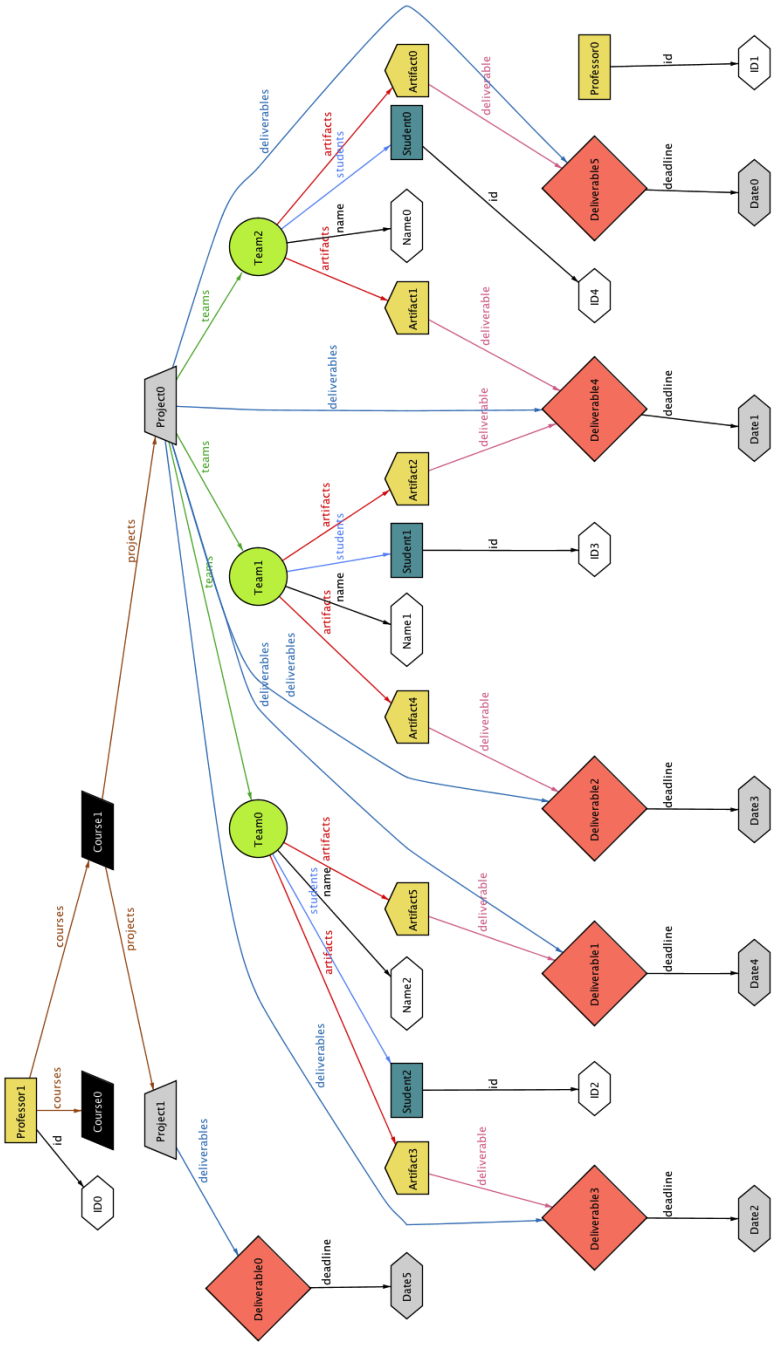

Figure 21: World # 1

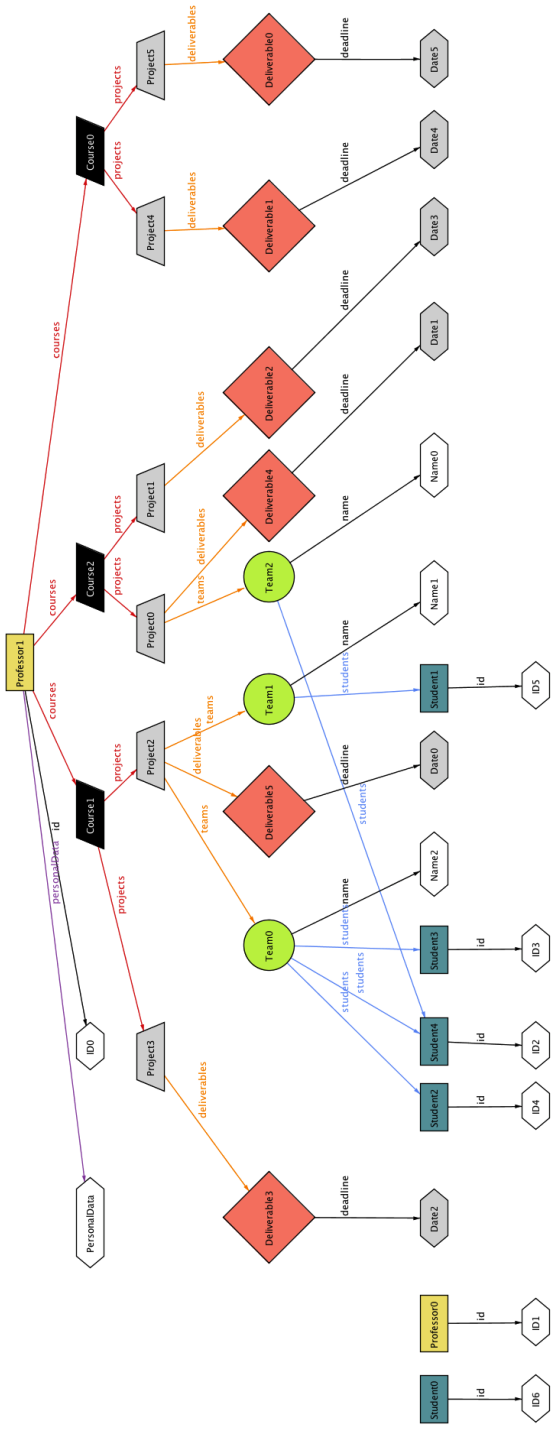

Figure 22: World # 2

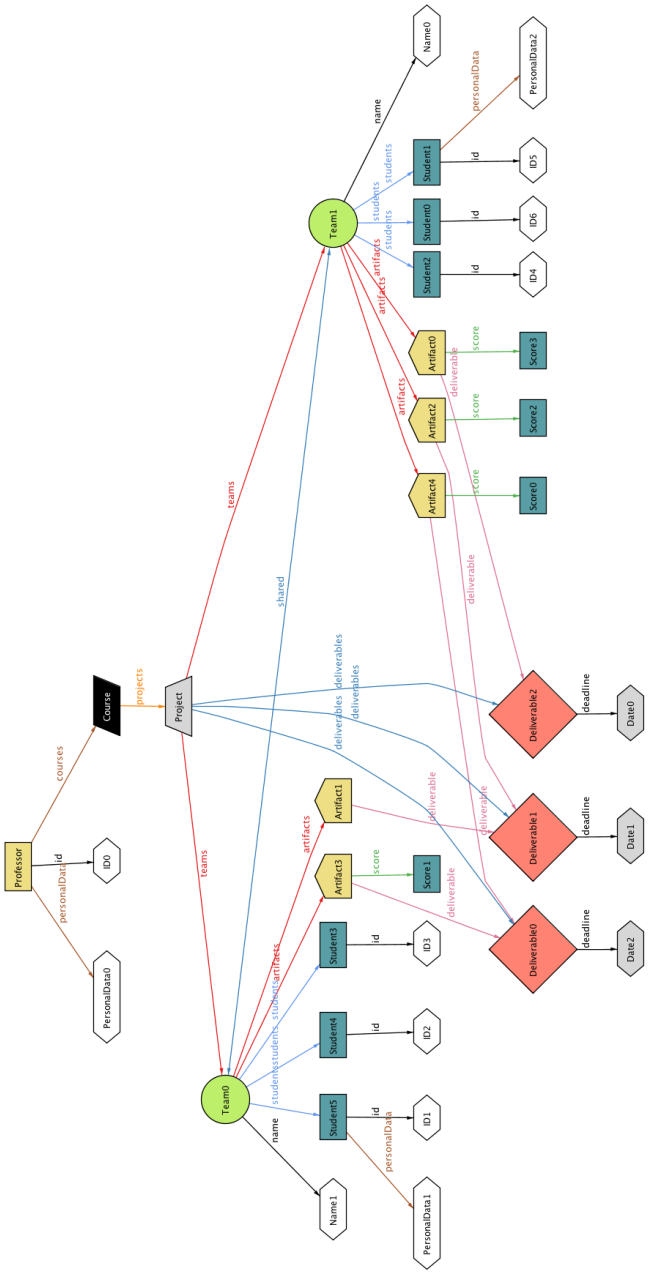

Figure 23: World # 3