Carleton University

cuPID System Requirement Analysis
Deliverable #1

Team Tech Support

Derek Felson
Margarita Otochkina
Christian Michael
Guelor Emanuel

Submitted to:

Dr. Christine Laurendeau
COMP3004

Due Date:
Wednesday, October 14, 2015

Contents

1. Introduction

1.1 Purpose of cuPID System

The *Carleton University Project Partner Identifier* (*cuPID*) system is designed to make it easier to match students into good teams for group project work. Many university courses include team projects, as they are a way to both teach students about working in groups and allow them to complete larger assignment than they could handle alone. Unfortunately, a common challenge is matching students into teams that are likely to work well together. Often when teams are formed, there is a mismatch between students' work habits, skillsets, or personalities, and such poorly formed teams tend to perform badly or break up before the project can be completed. The cuPID system aims to help prevent that by matching students into teams that they should be compatible with.

The system is intended to be used by Carleton students, faculty, and perhaps staff in the form of TAs. However, the system itself will only recognize three classes of users: students, administrators, and guests. The role of a guest is simply to create an administrator or student account.

The cuPID system has several features that work together to allow good team matches to be identified:

1. An administrator can create projects and edit their settings such as team size, name, and description.
2. A student can join any number of projects and create a profile for themselves, consisting of at least twelve qualifications relevant to team compatibility, such as grades, habits, skillsets, and personality. Along with each qualification, the profile contains a corresponding preference regarding a range of values the student would find acceptable in their team members. Students can later choose to edit their profile.
3. The administrator can select an option to compute the best teams for a project; the cuPID system will use the information in the profiles of students registered in that project to identify the most compatible teams. The administrator can then choose to view the results in either a summary form, which contains just a list of teams and which students are in them, or in detailed form, which contains specifics on why the teams were put together as they were.

To make administration easier when dealing with large class sizes, the system also allows guests to use the system without logging in, for the purpose of creating a new student (or administrator) account. Creating a student profile is part of the student account creation process.

1.2 Overview of Document

The purpose of this document is to explain the proposed high-level design of the cuPID system from the point of view of the user or client. This document is written in plain language based on the description provided by the client. It should be used by both the client and the developers to ensure that all parties can agree on what system will be built, what its main features are, and how it is to behave. Implementation details that are primarily of interest to developers, such as database design and menu navigation, are outside the scope of this document.

Because even a high-level description of a system as complex as cuPID must be understood on several levels, the next section breaks the proposed design into parts, beginning with a simple list of features and proceeding to cover the high-level aspects of the system in a variety of ways and at different levels of detail.

2. Proposed System

2.1 Overview

This documents breaks up the description of the proposed system design into five main sections:

1. Functional requirements: these identify the key features of the cuPID system.
2. Non-functional requirements: these are user-visible constraints on the design of the system and cover areas such as usability, reliability, performance, and supportability.
3. Use cases: this section includes both a written description of how every feature behaves, as well as diagrams that put those features into the context of the overall system--and each other, where necessary.
4. Object model: this section includes a data dictionary, which gives definitions to the major concepts and entities that exist in the proposed system, and diagrams showing how those entities (classes) relate to each other.
5. Dynamic model: these show in a more detailed and sequential fashion how each feature will behave over time.

2.2 Functional Requirements (FR)

A functional requirement is a user-visible feature of a system. In the context of cuPID, the functional requirements cover what features the client asked for in the project description as well as a few that are implied by it. Functional requirements for cuPID describe what a student can do, what an administrator can do, and what a guest user can do.

2.2.1 Student Functionality

F-01	Student users must be able to add themselves to an existing project
F-02	Student users must be able to edit their own project partner profile
F-03	Student users must be able to view a list of existing projects

2.2.2 System Administrator Functionality

F-04	Administrator users must be able to edit project settings
F-05	Administrator users must be able to assess Project Partner Compatibility
F-06	Administrator must be able to view the list of existing projects
F-07	Administrator users must be able to create a new project
F-08	Administrator users must be able to view Summary Result for the Match result
F-09	Administrator users must be able to view Detailed Result for the Match result

2.2.3 Guest Functionality

F-10	Guest users must be able to create a new accounts
F-11	Guest users must be able to create Student account
F-12	Guest users must be able to create Administrator account

2.3 Non-Functional Requirements (NFR)

A non-functional requirement is a user-visible constraint on the design of the system. They are typically broken down into four main categories (usability, reliability, performance, and supportability), as well as several others, in this case five. In cuPID, the non-functional requirements are informed by assumptions made regarding issues such as class size, who uses the system, and what the system may need to support in the future.

2.3.1 Usability

NF-01 The system navigation and interface should be intuitive so that average number of user mistakes is less than one for each high level use case

NF-02 Profile qualifications and preferences should be easy to determine so that 90% of students can create a profile in less than 10 minutes

NF-03 All language should be clear, descriptive, and professional, as agreed by 90% of surveyed users

NF-04 90% of surveyed users should agree that the look and feel of the system—including fonts, style, colour, etc.—is professional, like the UI of commercial products

NF-05 90% of surveyed users should agree that the level of user documentation provided for installation and operation of the system is adequate

NF-06 Keyboard shortcuts must be provided where applicable so that the interface can be operated without a mouse

2.3.2 Reliability

NF-07 The number of all possible failures like system crash or inability to save changed data during one-day usage must be less than 2

NF-08 The system must be able to process invalid user input avoiding system crash or corrupting data

NF-09 If the system fails during usage it must be able to load the last saved settings

NF-10 The system must be able to provide a stable and reliable connection between client and server

> **NF-10-01** The system must be able to notify user if during usage client gets disconnected and attempt to reconnect

2.3.3 Performance

NF-11 The system must be able to start in less than a minute on a computer with at least 2GB RAM or 4GB RAM if it is running on virtual machine

NF-12 The system must be able to store efficiently at least 1,000 student profiles without noticeably slowing down

NF-13 The program must be able to run the project partner matching algorithm for 100 profiles in less than a minute

NF-14 The system must perform profile/project information update in less than 10 seconds

NF-15 The system must be available all the time without any downtime

2.3.4 Supportability

NF-16 The System must be extensible to support email notification regarding students events on the system.

- **NF-16-01** Student must receive an email in case the project he/she joined has been deleted

NF-17 The system must be extensible to support the ability to have more than one student concurrently log in to the system

NF-18 The system must be extensible for mobile and tablet devices

NF-19 The system must be extensible to support passwords and password reset

- **NF-19-01** Users must be able to reset password via web interface
- **NF-19-02** Users must be able to reset password using their student email address

2.3.5 Implementation

NF-20 The system must run on the COMP 3004 Ubuntu Linux virtual machine

NF-21 All source code must be written in C++

NF-22 The system must have a graphical interface

2.3.6 Packaging

NF-23 The client must be able to download the source tar file from official website

NF-24 Installation of the application/program must not take more than 5 mins

- **NF-24-01** Installation may take more time if the system requirements doesn't meet the requirements of the application
- **NF-24-02** Installation may be cancelled due to limited amount of free storage on the device/computer

NF-25 There must be only 1 installation of the system on a single computer

2.3.7 Interface

NF-26 Because the client may eventually want different programs to analyze students and teams in different ways, the system must provide a general API for future authorized programs to access its data independently of any cuPID features or the provided user interface.

NF-27 Because the client may eventually want to support network access to cuPID from remote hosts in a future version, the system must provide an API to allow access to all cuPID features, independently of the user interface the system is delivered with.

NF-28 Because analysis and consultation with the client identified web access to cuPID as a possible future feature, any API provided for accessing cuPID must be designed according to best practices for RESTful web services.

2.3.8 Legal

NF-29 We will not be liable/responsible for any system failures or damages

NF-30 The system must comply by the University Collection of Personal Information privacy policy

NF-31 There won't be any specific components or third party libraries incorporated into the system in order to avoid any form of royalties fee. Any library used will be provided by the operating system itself

2.3.9 Operations

NF-32 The system must have schedule updates to keep the student private information secured

NF-32-01 The user must be able to download updates for the system online from the official website

NF-33 The system must have a forum or website where users can post issues regarding the system, and request for features they would like to see

NF-34 The Client Organization is responsible for maintaining the system and any end-user support after two years of deployment

2.4 System Model

A result of requirements elicitation and analysis, a system model is (1) a specification of the features of a system from the user's point of view and (2) a precise, high-level model of system behaviour, object behaviour, and object interactions. It is based on a system's functional and non-functional requirements.

In this document, the system model includes three types: the use case model, the object model, and the dynamic model. The use case model specifies the features of the system from the user's point of view. The object model defines what objects are in the system as well as their behaviour, relationships, and interactions. The dynamic model describes the system behaviour. These work together to provide a high-level model and specification of the system as a whole.

2.4.1 Use Case Model (UC)

Based on the functional (and non-functional) requirements of a system, a use case model is a way of defining a system's features and requirements in a clear and unambiguous way based on how the users can interact with it. Each feature identified in the functional requirements is some way the system can be used, and use cases are an abstraction for capturing those user-system interactions. In addition, use cases can describe error cases that may arise from other use cases.

This section is organized into two parts. The first part is an overview of the use cases identified in cuPID. This includes use case diagrams that show how every use case works together to provide all the functionality in the system, and tables including the names and short descriptions of all use cases. The second part is the use case flow of events, which describes every use case in detail.

A. High-Level Use Case Diagram

This section contains tables to identify and briefly describe all use cases and diagrams to show the relationships between them.

Diagram 1 shows the overall functionality of the system in terms of five high-level use cases and the three actors that initiate them.

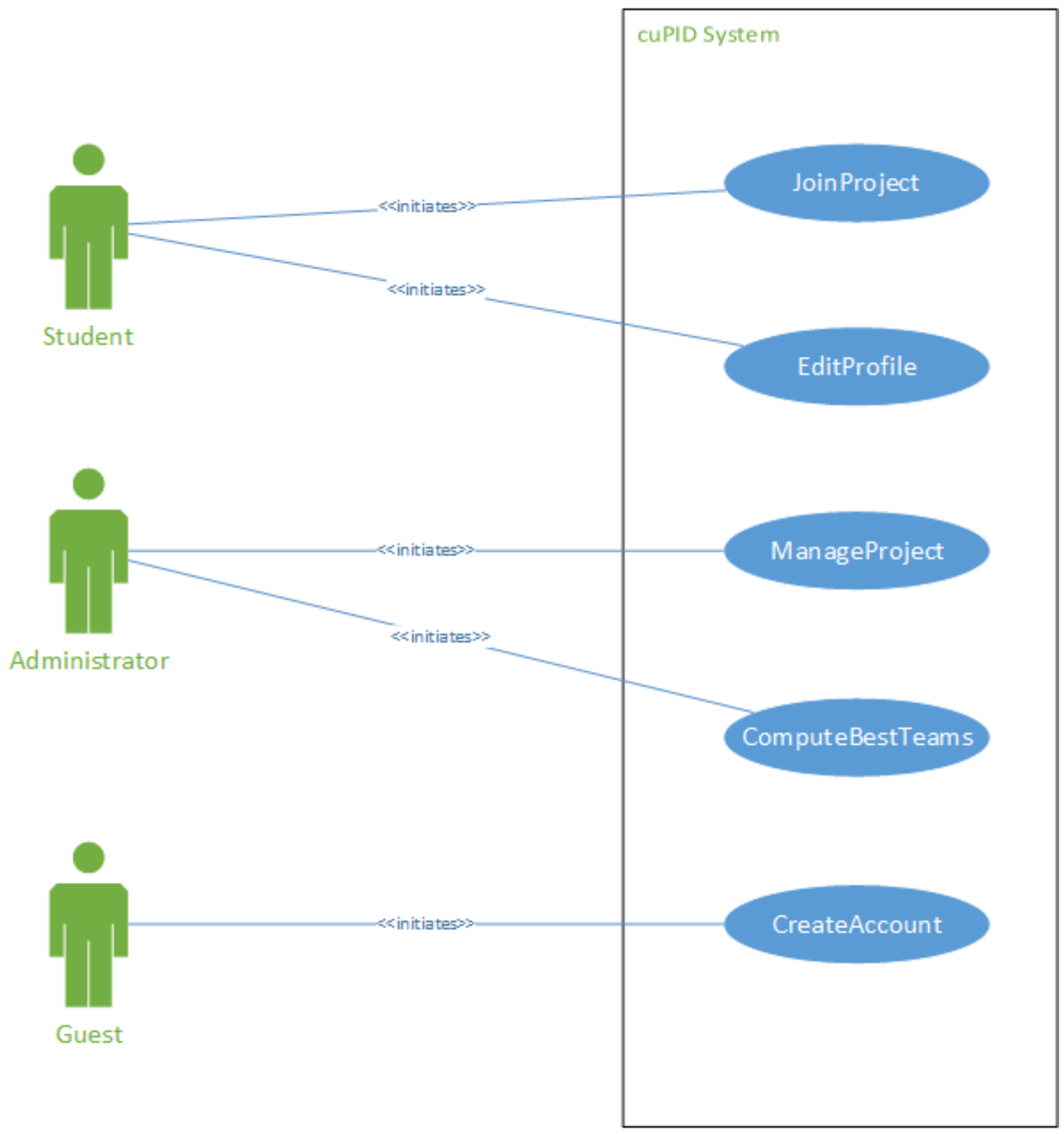

Diagram UCD-01: High-level Use Cases

B. High-Level Use Case Description

Table 1 briefly describes each of the five high-level use cases in Diagram 1.

Table 1: High-level Use Case Descriptions

UC #	Case Name	Case Description
UC-01	CreateAccount	Guest user can create new account: new Student

		account or new Administrator account.
UC-02	EditProfile	Student can edit his/her cuPID profile.
UC-03	JoinProject	Student can view available projects and apply to any of them.
UC-04	ManageProject	Administrator can view all created projects and create a new one.
UC-05	ComputeBestTeams	Administrator can run the Project Partner Matching algorithm and view the result.

C. Detailed Use Case Diagrams

Of those five high-level use cases, three include other cases. The following diagrams display the relationships between those use cases in detail.

ManageProject is a high-level use case where an Administrator creates or edits a project. To reduce the complexity of UCD-01, these two similar interactions were combined into one high-level use case, though they represent distinct features.

Diagram UCD-02 shows how ManageProject is related to other use cases.

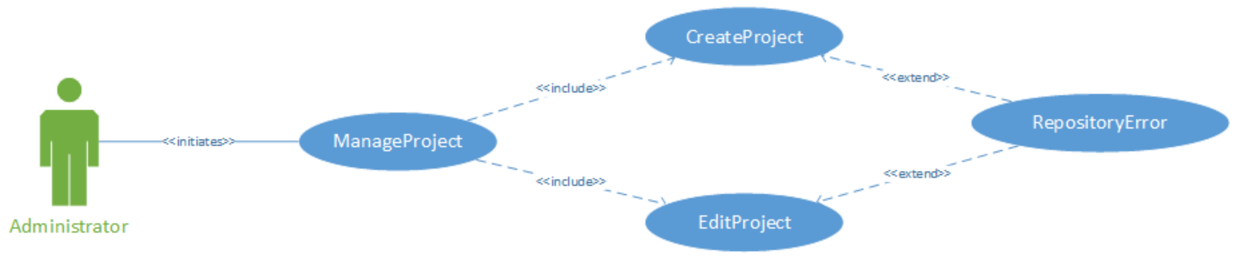

Diagram UCD-02: ManageProject Detailed Use Case

ComputeBestTeams is a high-level use case that covers an administrator telling the cuPID system to find the best teams for a project and reviewing the results. An administrator can opt to view the results in either summary or detailed form.

Diagram UCD-03 shows how ComputeBestTeams is related to other use cases.

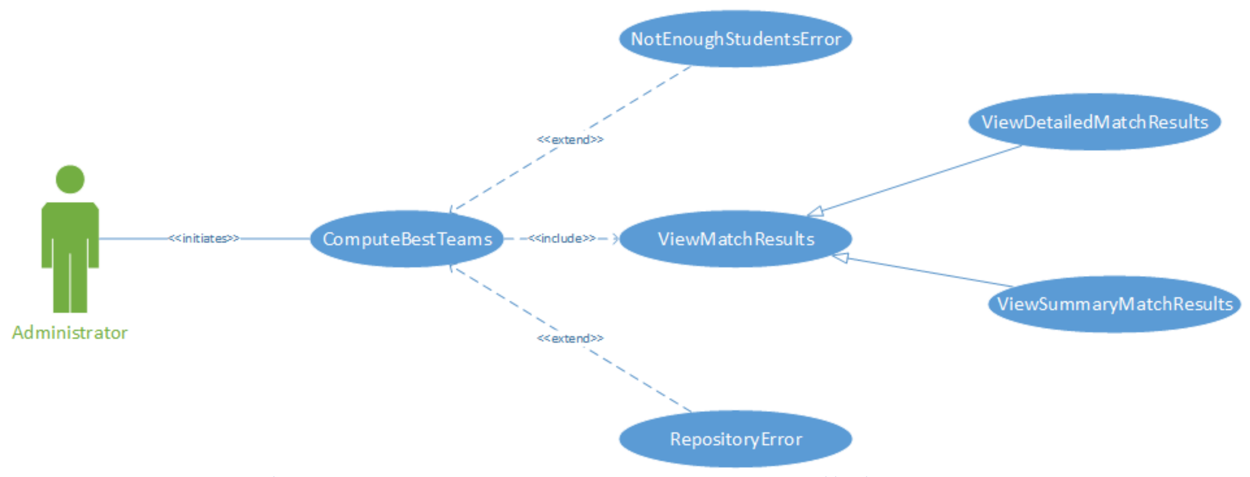

Diagram UCD-03: ComputeBestTeams Detailed Use Case

CreateAccount is a high-level use case that covers the three features usable by a Guest: the creation of student and administrator accounts, and creating a student profile.

Since one cannot create a student account without creating a profile, CreateStudentAccount includes CreateProfile; they are shown as two use cases to prevent the CreateStudentAccount use case from becoming too complex and to make it clearer that the feature of creating a profile is represented in the system by a use case.

CreateAdminAccount and CreateStudentAccount were combined into CreateAccount to reduce the complexity of Diagram UCD-01. Various errors may result from invalid user input at any step during the account creation process; each use case except CreateAdminAccount can produce a different type of error.

Diagram 4 shows how CreateAccount is related to other use cases.

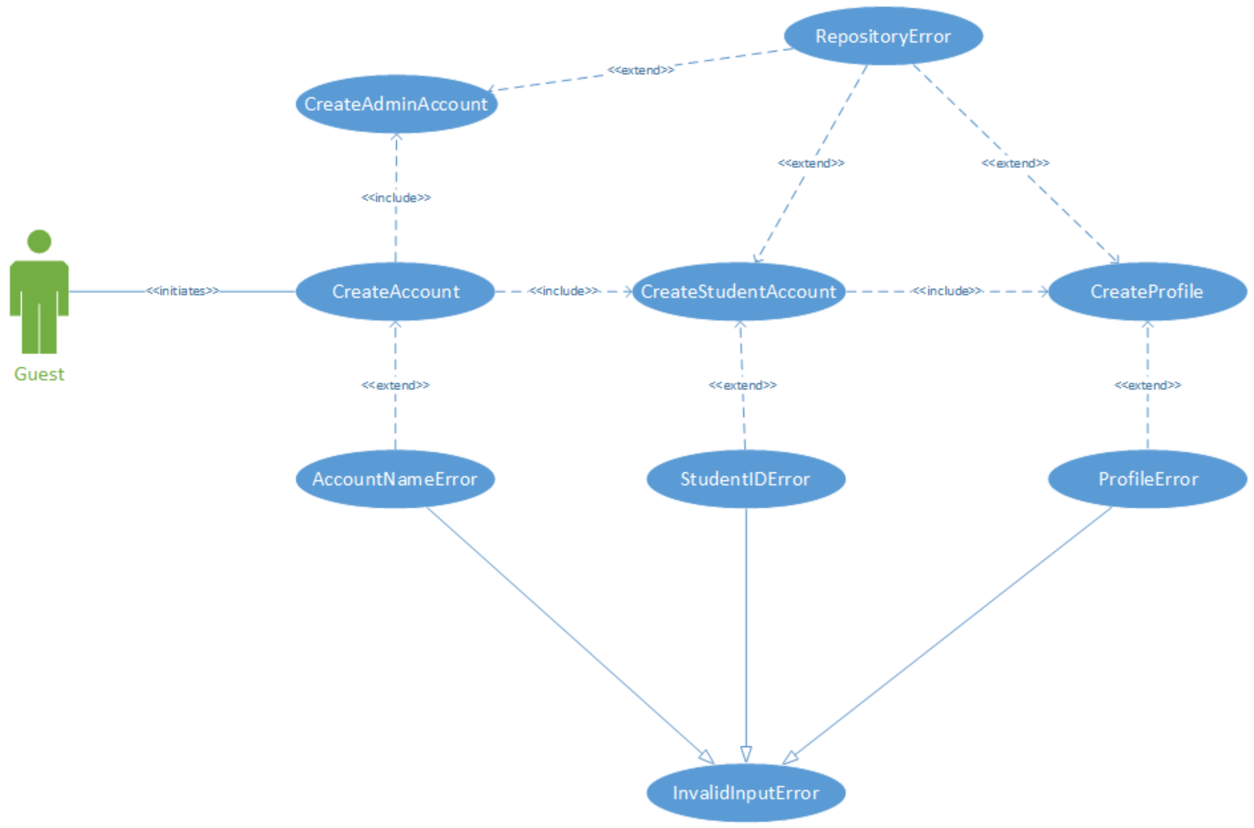

Diagram UCD-04: CreateAccount Detailed Use Case

The JoinProject use case is fairly simple, with only a single error case if there is a problem with the repository. Diagram UCD-05 shows the relationship.

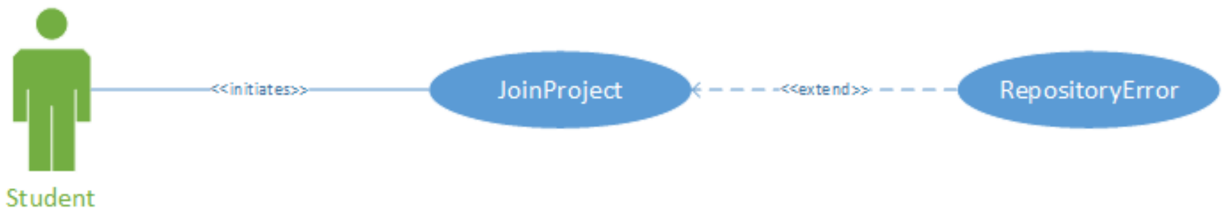

Diagram UCD-05: JoinProject Detailed Use Case

The EditProfile use case and its relationships to other use cases is shown in Diagram UCD-06.

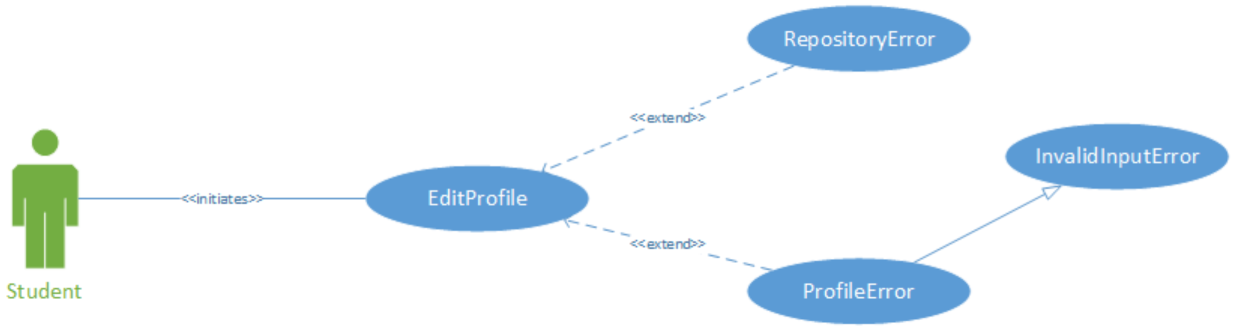

Diagram UCD-06: EditProfile Detailed Use Case

D. Detailed Use Case Description

Table 2 briefly describes all the use cases in the system, except those five high-level use cases already described in Table 1.

Table 2: Detailed Use Case Descriptions

UC #	UC Name	UC Description
UC-06	CreateAdminAccount	Guest user creates new Administrator account.
UC-07	CreateStudentAccount	Guest user creates new Student account.
UC-08	CreateProfile	Guest user creates profile for Student profile.
UC-09	CreateProject	Administrator user creates new project and make it available for Student users.
UC-10	EditProject	Administrator user edits selected project's settings.
UC-11	ViewSummaryResult	Administrator user views Summary result after running Project Partner Matching algorithm.
UC-12	ViewDetailedResult	Administrator user views Detailed result after running Project Partner Matching algorithm.
UC-13	ViewExistingProjects	Registered user views all of the existing projects.
UC-14	AccountNameError	The System reports that the account name given by Guest user is invalid.
UC-15	StudentIDError	The System reports that the Student ID number given by Guest user is invalid.
UC-16	ProfileError	The System reports an error during Profile information processing.
UC-17	InvalidInputError	The System reports an error during processing of entered information.
UC-18	NotEnoughStudentsError	The System informs the Administrator user that the number of students joined the Project is too small.
UC-19	RepositoryError	The system notifies the Guest/User/Administrator that there was a problem loading or saving the request.

E. Use Case Flow of Events

This section provides details on all the use cases in the system.

Because almost all use cases require the user to be logged in as a student or administrator, to reduce redundancy the entry conditions do not always explicitly state this. Instead, if an entry condition includes something such as "Administor elects to..." (or "Student elects to..."), that implies that the user must be logged in as an administrator (or student).

The CreateAccount use case allows the Guest to create a new student or administrator account, perhaps so that they can then use the system in one of those roles.

Table UCF-01: Use Case Flow for CreateAccount High-level Use Case

Use Case Identifier	**UC-01**
Name	**CreateAccount**
Participating Actor	**Initiated by Guest**
Flow of Events	• Guest user makes a choice between two options: ○ Create Student Account (**CreateStudentAccount** case) ○ Create Administrator account (**CreateAdminAccount** case) • System receives Guest user's choice.
Entry Condition	Guest user has started Registration process.
Exit Condition	System transfers Guest user to profile creation of his/her choice.
Quality Requirement	
Traceability	F-10, F-11, F-12, NF-01

The EditProfile use case allows the Student to make changes to the qualifications and preferences in their own profile.

Table UCF-02: Use Case Flow for EditProfile High-level Use Case

Use Case Identifier	**UC-02**
Name	**EditProfile**
Participating Actor	Initiated by Student
Flow of Events	• Student user edits desired fields and select the save option. ○ The System updates Student's most recent information by sending new data to repository and returns a message to Student indicating if his information was saved successfully. • Student receives Success message.
Entry Condition	Student user has elected to edit profile.
Exit Condition	Student user's information has been saved and updated.
Quality Requirement	• During profile information update the System displays a message to Student asking to wait till the end of update.

	• New information update must be done in less that 5 seconds. • If Student user enters invalid information during account creation, System must notify the user about it and show suggested solution to a problem.
Traceability	F-02, NF-01, NF-02, NF-06, NF-07, NF-08, NF-09, NF-14

The JoinProject use case allows the Student to register with an existing project.

Table UCF-03: Use Case Flow for JoinProject High-level Use Case

Use Case Identifier	**UC-03**
Name	**JoinProject**
Participating Actor	Initiated by Student
Flow of Events	• The Student user selects one of the existing projects. ○ The System displays a project to the user. • The Student user selects a project to join if option is available. ○ The System receives the request, adds the user to the list of Students joined the project and sends Confirm message to the user. • Student user confirms the action.
Entry Condition	Student user has elected to view existing projects.
Exit Condition	Student user has been added to the project.
Quality Requirement	• The System should not present an option to join a project the Student has already been added to.
Traceability	F-01, F-03, NF-14, NF-16-01

The ManageProject use case allows an administrator to create a new project or edit an existing one.

Table UCF-04: Use Case Flow for ManageProject High-level Use Case

Use Case Identifier	**UC-04**
Name	**ManageProject**
Participating Actor	Initiated by **Administrator**
Flow of Events	• The Administrator user selects View Projects option. ○ The System list all available projects. • If Administrator user decides to create new Project(**createProject** case), he/she selects Create New Project option.

	○ The System starts **createProject** process. • If Administrator user decides to edit existing Project(**editProject** case), he/she selects one of the projects. ○ The System displays Project page. • Administrator user selects Edit Project option. ○ The System starts **EditProject** process.
Entry Condition	Administrator must be logged into the System.
Exit Condition	The System starts selected process.
Quality Requirement	• Saving and loading processes must not take more than 1 minute. • The menu system should be very responsive.
Traceability	F-04, F-06, F-07, NF-01, NF-06, NF-07, NF-08, NF-14

The ComputeBestTeams use case allows the administrator to find the optimal configuration of students to teams for a project and review the results in either summary or detailed form.

Table UCF-05: Use Case Flow for ComputeBestTeams High-level Use Case

Use Case Identifier	**UC-05**
Name	**ComputeBestTeams**
Participating Actor	Initiated by Administrator
Flow of Events	• Administrator user selects one of the Projects. ○ The System displays Project page to the user. • Administrator user selects Partner Matching option. ○ The System checks all of the project requirements, runs the Algorithm, compute the best set of teams for the project and prints computed information to the user. • Administrator receives Summary and Detailed information.
Entry Condition	Administrator user has elected to run Partner Matching Algorithm for one of the created Projects.
Exit Condition	
Quality Requirement	• If the number of Students applied to the Project is to small for running the algorithm, System must notify the user and suggest either to wait for more people to Join the Project or change Project settings. • If something went wrong during the team computation, System must notify the user about it and suggest the solution for the problem.
Traceability	F-05, F-06, F-08, F-09, NF-07, NF-09, NF-13

The CreateAdminAccount use case allows the Guest to create a new administrator account.

Table UCF-06: Use Case Flow for CreateAdminAccount Use Case

Use Case Identifier	**UC-06**
Name	**CreateAdminAccount**
Participating Actor	Initiated by **Guest**
Flow of Events	• Guest user fills in required fields in account creation form and submits the information. ○ System creates new account, saves entered information in it and sends Success message to Guest user.
Entry Condition	Guest user has elected to create new account and chose to create an Administrator account.
Exit Condition	New Administrator account has been created and saved in a System.
Quality Requirement	• If Guest user cancels the account creation the new account will not be created and saved. • If Guest user enters invalid information during account creation, System must notify the user about it and show suggested solution to a problem.
Traceability	F-10, F-12, NF-01, NF-02, NF-07, NF-08, NF-14

The CreateStudentAccount use case allows the Guest to create a new student account.

Table UCF-07: Use Case Flow for CreateStudentAccount Use Case

Use Case Identifier	**UC-07**
Name	**CreateStudentAccount**
Participating Actor	**Initiated by Guest**
Flow of Events	• Guest user enters his/her Student number and name. ○ System accepts Student's number and name and transfers him/her to Student creation page. • Guest user fills in all the required fields on a page and submits the information. ○ System accepts the information, creates new Student account, adds entered information to it and sends Success message to the user. • Guest user receives Success message.
Entry Condition	Guest user has elected to create new account and chose to create an Student account.
Exit Condition	New Student account has been created and saved in a System.
Quality Requirement	• If Guest user cancels the account creation the new account will not be created and saved. • If Guest user enters invalid information during account creation, System must notify the user about it and show suggested solution to a problem. • If Student profile with given Student number/name already exists, System must notify the user about it and suggest to use

	another number/name.
Traceability	F-10, F-11, NF-01, NF-07, NF-08

The CreateProfile use case allows a Guest to create a profile (including qualifications and preferences) as part of creating a new student account.

Table UCF-08: Use Case Flow for CreateProfile Use Case

Use Case Identifier	**UC-08**
Name	**CreateProfile**
Participating Actor	Initiated by Guest
Flow of Events	• Guest user selects Create Student Profile option. o System accepts the request and displays Profile form to the user. • Guest user fills in all the required fields on a page and submits the information. o System accepts the information, creates new Student account, adds entered information to it and sends Success message to the user. • Guest user receives Success message.
Entry Condition	Guest user has elected to create new profile.
Exit Condition	New profile has been created and added to the system.
Quality Requirement	
Traceability	NF-13, NF-01, NF-02, NF-07, NF-08, NF-09, NF-12, NF-14

The CreateProject use case allows an Administrator to make a new project and define its properties such as title, description, and team size.

Table UCF-09: Use Case Flow for CreateProject Use Case

Use Case Identifier	**UC-09**
Name	**CreateProject**
Participating Actor	Initiated by Administrator
Flow of Events	• Administrator user enters name for a new Project and presses Next. o The System accepts Project name, creates new Project and opens Project creation page for the user. • Administrator user enters Project description and settings and submits entered information. o The System receives Project information, saves it and sends Success message to the user. • Administrator user receives Success message.
Entry Condition	Administrator user has elected to create new Project.

Exit Condition	New Project has been created and saved in the System.
Quality Requirement	• If Administrator user enters invalid Project information, the System must notify the user and show suggested solution to a problem.
Traceability	F-07, NF-01, NF-07, NF-08, NF-14

The EditProject use case allows an Administrator to modify the settings of an existing project, such as title, description, and team size.

Table UCF-10: Use Case Flow for EditProject Use Case

Use Case Identifier	**UC-10**
Name	**EditProject**
Participating Actor	Initiated by Student
Flow of Events	• Student user opens his/her profile and chooses Edit Profile option. ○ System makes Student's profile page editable. • Student changes desired fields and submits changes. ○ System receives new information, updates Student's profile and sends Success message to the user. • Student user receives Success message.
Entry Condition	Student user has elected to edit his/her profile.
Exit Condition	New Profile information has been received and updated in a System.
Quality Requirement	• If Student user cancels the Profile edition, non of the changed information will be saved. • If Student user leaves some filed(s) blanc, System must notify the user that empty fields are not allowed and point to the empty fields. • If Student user enters invalid Profile information, System must notify the user and show suggested solution to a problem.
Traceability	F-04, F-06, NF-01, NF-07, NF-08, NF-09, NF-14

The ViewSummaryResult use case allows the Administrator to see a simple list of the best computed teams (and the students in them) for a project.

Table UCF-11: Use Case Flow for ViewSummaryResult Use Case

Use Case Identifier	**UC-11**
Name	**ViewSummaryResult**
Participating Actor	Initiated by Administrator
Flow of Events	• Administrator user has received Partner Matching Result for the Project and chooses to view Summary Result.

	○ System displays names of all Students joined the Project grouped by teams.
Entry Condition	Administrator user has elected to run Partner Matching Algorithm for one of the created Projects.
Exit Condition	
Quality Requirement	
Traceability	F-08, NF-03

The ViewDetailedResult use case allows the Administrator to see a detailed list of the best computed teams (and the students in them) for a project, as well as the reasons why those students were matched to those teams.

Table UCF-12: Use Case Flow for ViewDetailedResult Use Case

Use Case Identifier	**UC-12**
Name	**ViewDetailedResult**
Participating Actor	Initiated by Administrator
Flow of Events	• Administrator user has received Partner Matching Result for the Project and chooses to view Detailed Result. ○ System displays rules and data supporting how and why the teams in Summary Result information have been computed.
Entry Condition	Administrator user has elected to run Partner Matching Algorithm for one of the created Projects.
Exit Condition	
Quality Requirement	
Traceability	F-09, NF-03

The ViewExistingProjects use case allows a Student or Administrator to see a list of existing projects in the system.

Table UCF-13: Use Case Flow for ViewExistingProjects Use Case

Use Case Identifier	**UC-13**
Name	**ViewExistingProjects**
Participating Actor	Initiated by Student, Administrator
Flow of Events	• Registered user selects View Projects option in a menu. ○ The System loads the list of all the existing projects and displays it to the user.
Entry Condition	Registered user must be logged into the System.
Exit Condition	
Quality Requirement	• The System must be able to list all of the created and opened to Students Projects. • Loading process must not take more than 1 minute.

Traceability	F-03, F-06, NF-03, NF-12

The AccountNameError use case allows the system to handle the error case in CreateAccount where the Guest provides an account name that is invalid or is already in use by another user account in the system.

Table UCF-14: Use Case Flow for AccountNameError Use Case

Use Case Identifier	**UC-14**
Name	**AccountNameError**
Participating Actor	Guest
Flow of Events	• System notifies the user that the name he entered for profile has already been used for another profile or invalid.
Entry Condition	Accept account name and number operation is aborted.
Exit Condition	Student account creation operation is aborted.
Quality Requirement	
Traceability	NF-01, NF-07, NF-08, NF-09

The StudentIDError use case allows the system to handle the error case in CreateStudentAccount where the Guest provides a student ID that is invalid or already being used by another student account in the system.

Table UCF-15: Use Case Flow for StudentIDError Use Case

Use Case Identifier	**UC-15**
Name	**StudentIDError**
Participating Actor	Guest
Flow of Events	• System notifies the user that the student ID he entered for profile has already been used for another profile or invalid.
Entry Condition	Accept account name and number operation is aborted.
Exit Condition	Student account creation operation is aborted.
Quality Requirement	
Traceability	NF-01, NF-02, NF-07, NF-08, NF-09

The ProfileError use case allows the system to handle the error case in CreateProfile where the Guest provides invalid information for the qualifications or preferences.

Table UCF-16: Use Case Flow for ProfileError Use Case

Use Case Identifier	**UC-16**
Name	**ProfileError**
Participating Actor	Student

Flow of Events	• System notifies the user that the error has occurred with entered information processing during account creation/editing.
Entry Condition	• Student left some fields blanc during profile creation/editing. • Student used invalid information for his/her profile.
Exit Condition	Student profile editing/creation is aborted.
Quality Requirement	
Traceability	NF-01, NF-02, NF-07, NF-08, NF-09

The InvalidInputError use case allows the system to handle generic error cases where a Student, Administrator, or Guest provides invalid input in another use case.

Table UCF-17: Use Case Flow for InvalidInputError Use Case

Use Case Identifier	**UC-17**
Name	**InvalidInputError**
Participating Actor	Student, Administrator, Guest
Flow of Events	• System notifies the user that the error has occurred with entered information processing or some fields have been left blanc during account the process.
Entry Condition	Information procession operation is aborted.
Exit Condition	• Account creation is aborted • Account edition is aborted • Profile creation/edition is aborted.
Quality Requirement	
Traceability	NF-01, NF-02, NF-07, NF-08, NF-09

The NotEnoughStudentsError allows the system to handle the error case in ComputeBestTeams where the Administrator attempts to compute the teams for a project where there are no students registered or it is not possible to meet the minimum size requirement of all teams given the number of students and the project settings.

Table UCF-18: Use Case Flow for NotEnoughStudentsError Use Case

Use Case Identifier	**UC-18**
Name	**NotEnoughStudentsError**
Participating Actor	Administrator
Flow of Events	• System notifies the Administrator user that the number of Students joined the project is to small for running the algorithm. The system suggests to wait for more students to join the project or to change Project Settings.
Entry Condition	Matching Partner algorithm is stopped.
Exit Condition	Matching Partner algorithm is aborted.

Quality Requirement	
Traceability	NF-07, NF-13

The RepositoryError use case allows the system to handle generic error cases stemming from issues related to the data repository, such as reading data, writing data, opening the repository, or data corruption.

Table UCF-19: Use Case Flow for RepositoryError Use Case

Use Case Identifier	**UC-19**
Name	**RepositoryError**
Participating Actor	Administrator, Student, Guest
Flow of Events	• System notifies the user that some error occurred during retrieving/saving/updating data from Repository.
Entry Condition	Current process is stopped.
Exit Condition	Current process is aborted.
Quality Requirement	
Traceability	NF-07, NF-12, NF-13, NF-14

2.4.2 Object Model

The purpose of an object model is to show the types of objects a system has to deal with and the relationships and associations between them. According to the cuPID project description provided by the client, the system has to work with student and administrator user objects and several kinds of other system entity objects, such as team, project, and profile. Further analysis revealed the importance of more user and system entity objects and relationships between them.

This section has three parts: the user object diagram, the system object diagram, and the data dictionary. The data dictionary defines each object and includes a list of its attributes.

A. User Entity Object Diagram

The cuPID project description mentions only two types of user: Student and Administrator. Analysis showed that a third type of user, Guest, was necessary for the creation of Student and Administrator accounts. Since these were all types of users, these objects inherit from the User object. What makes Students and Admistrators different from Guests is that Student and Administrators are registered with the system; therefore Student and Administrator also inherit from RegisteredUser, which is of course also a type of User.

Diagram OM-01 shows these relationships.

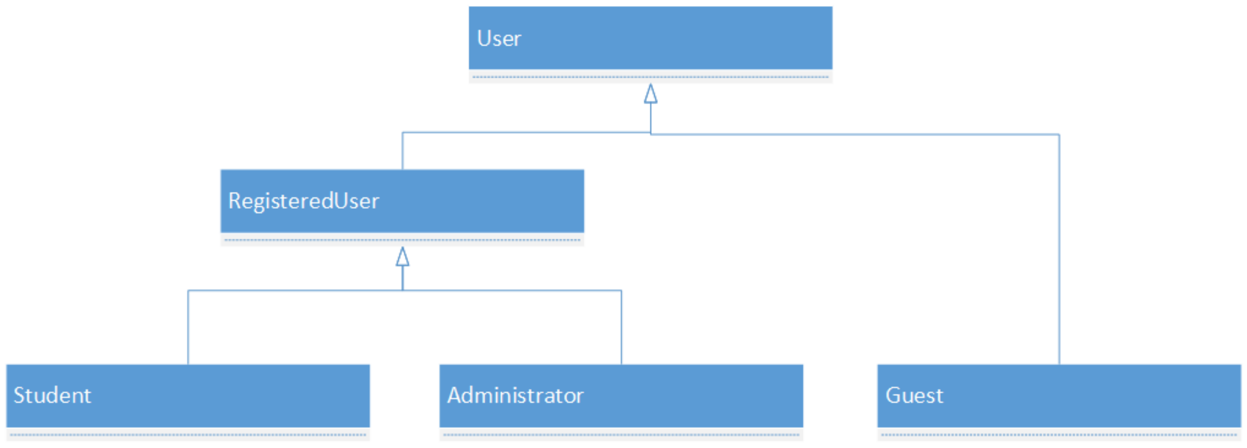

Diagram OMD-01: Entity Diagram for User Objects

B. System Entity Object Diagram

The cuPID project description provided by the client obviously mentions several types of system entity objects: profiles, projects, teams, and qualifications. Diagram OM-02 shows these objects and several others.

Careful analysis also showed that the project description includes language regarding the computed best teams for a project, which this document calls MatchResults.

In addition, the project description mentions viewing summary and detailed information about the computed matches, and therefore the proposed system includes system entity objects for that type of information as well. Analysis suggested that both DetailedMatchInformation and SummaryMatchInformation are specializations of the same kind of object, which the proposed system calls MatchInformation.

While the client-provided project description does not mention preferences, analysis of the profile object and its associated qualifications revealed that a profile contains both qualifications and some other information, which the proposed system calls preferences. Due to limitations in the notation, it was not possible to show the relationship between qualifications and preferences, but every qualification in a profile has exactly one corresponding preference, specifying the range of values for that qualification that the student would find acceptable in their team mates.

The Student object in this diagram is the same as the Student object in Diagram OM-01. Its relationships are omitted here to reduce complexity.

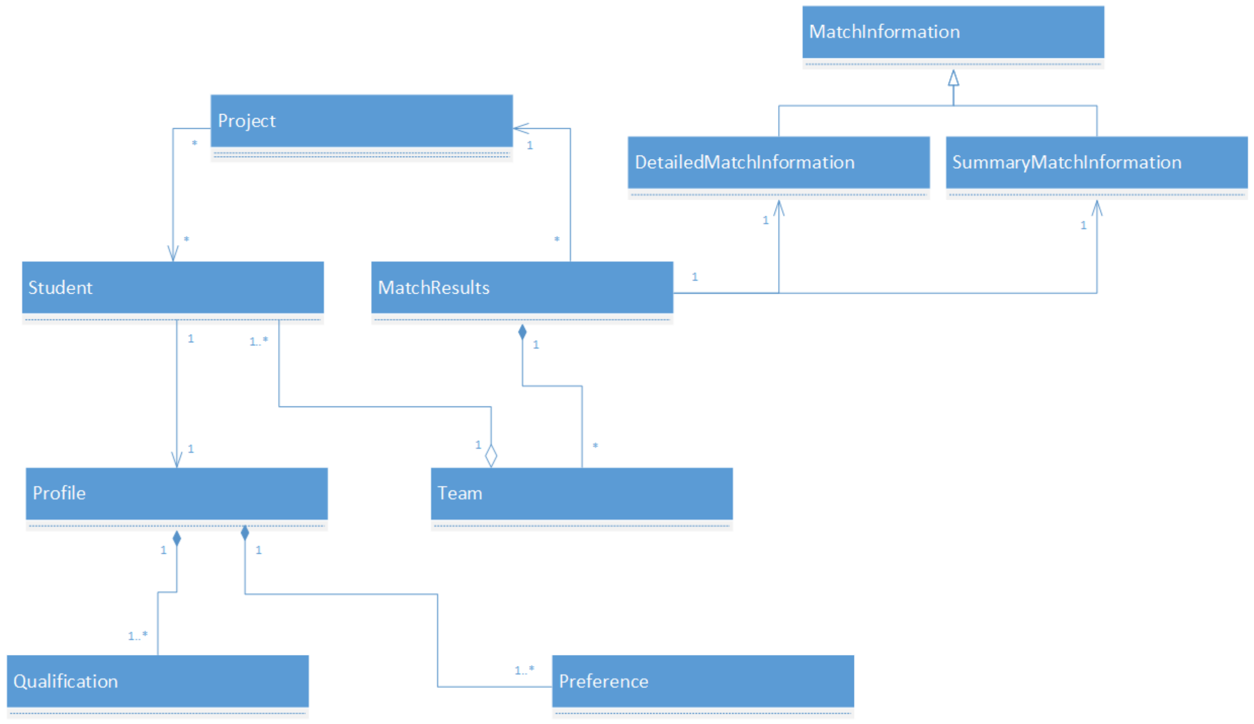

Diagram OMD-02: Entity Diagram for System Objects

C. Entity Object Data Dictionary

Table OMT-01 defines and lists the attributes of every object in the object model.

Table OMT-01: Data Dictionary

Entity Object	Attributes and Associations	Definition
User		A *Student*, *Administrator,* or another user that can access the cuPID system.
Student	studentId studentName profile	A user who is enrolled in a college or University, and participates in a *Project*. A student creates, delete, modify his/her own Profile/Account, and can join any *Project* that is available.
Administrator		The user class associated with advanced permissions.

Guest		The user class that is only able to create an account.
RegisteredUser	username	The user class with user name who is able to access the cuPID system.
MatchResult	summaryInformation detailedInformation teams	Data model object which contains information about the final result of Student's matching algorithm.
Project	id name minTeamSize maxTeamSize description registeredStudents	A set of requirement set by the *Administrator* for the purpose of grouping student into a team. A Project is always initiated by the *Administrator,* A *Project* is composed of a number of *Students*, a name, a description, and a status.
Team	teamMembers size	Data model object with at least one Student user class object chosen by algorithm.
Profile	qualifications preferences	A profile describes a Student's qualifications and preferences relevant to matching them to a team.
Qualification	name value preference	Part of a Profile, a Qualification contains information about some aspect of the student's own skill set, work habits, etc.
Preference	name minValue maxValue	A preference indicates the corresponding rage of values, for some Qualification, that a student is willing to accept in their teammates.
MatchInformation		An object containing information about the best teams as computed by the cuPID system for a project.
SummaryMatchInformation	teams	An object containing summary information about the computed best teams, specifically just a list of

		which students were matched to which teams.
DetailedMatchInformation	teams reasons	An object containing detailed information about the computed best teams, including both a list of students in each team and the reasons why they were put together.

2.4.3 Dynamic Model

A dynamic model describes the behaviour of a system by examining how individual use cases occur over time and how the state of stateful objects changes over time. It shows the former using UML sequence diagrams and the latter using UML state machine diagrams. However, there are no state machine diagrams here, since cuPID contains no entity classes that have state, and showing the state of boundary and control objects is outside the scope of this document.

This section has only one part, which includes the sequence diagrams for every use case.

A. Sequence Diagrams

The CreateAccount use case allows the Guest to create a new student or administrator account, perhaps so that they can then use the system in one of those roles.

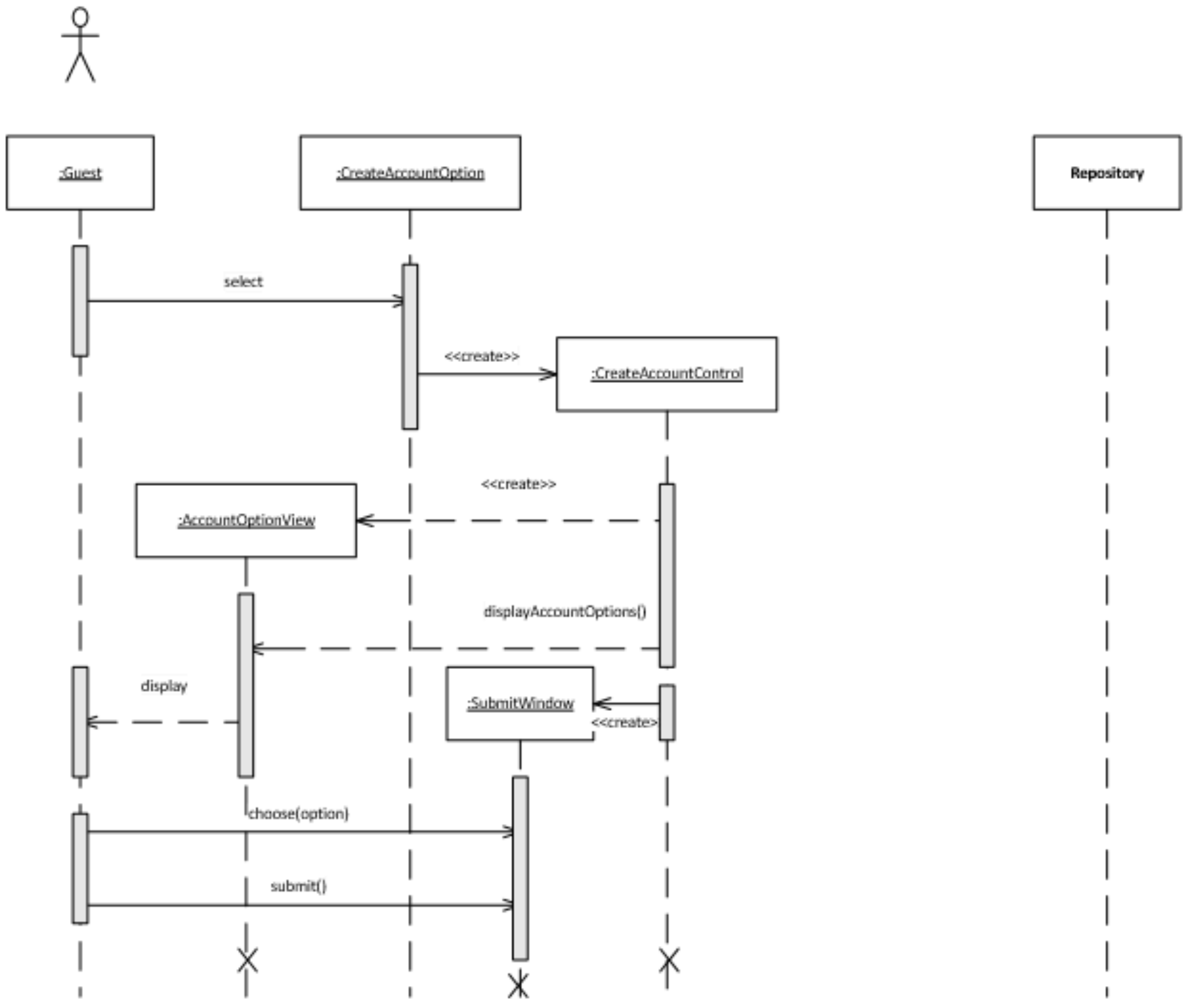

Diagram S-01: Sequence diagram for CreateAccount use case

The CreateAdminAccount use case allows the Guest to create a new administrator account.

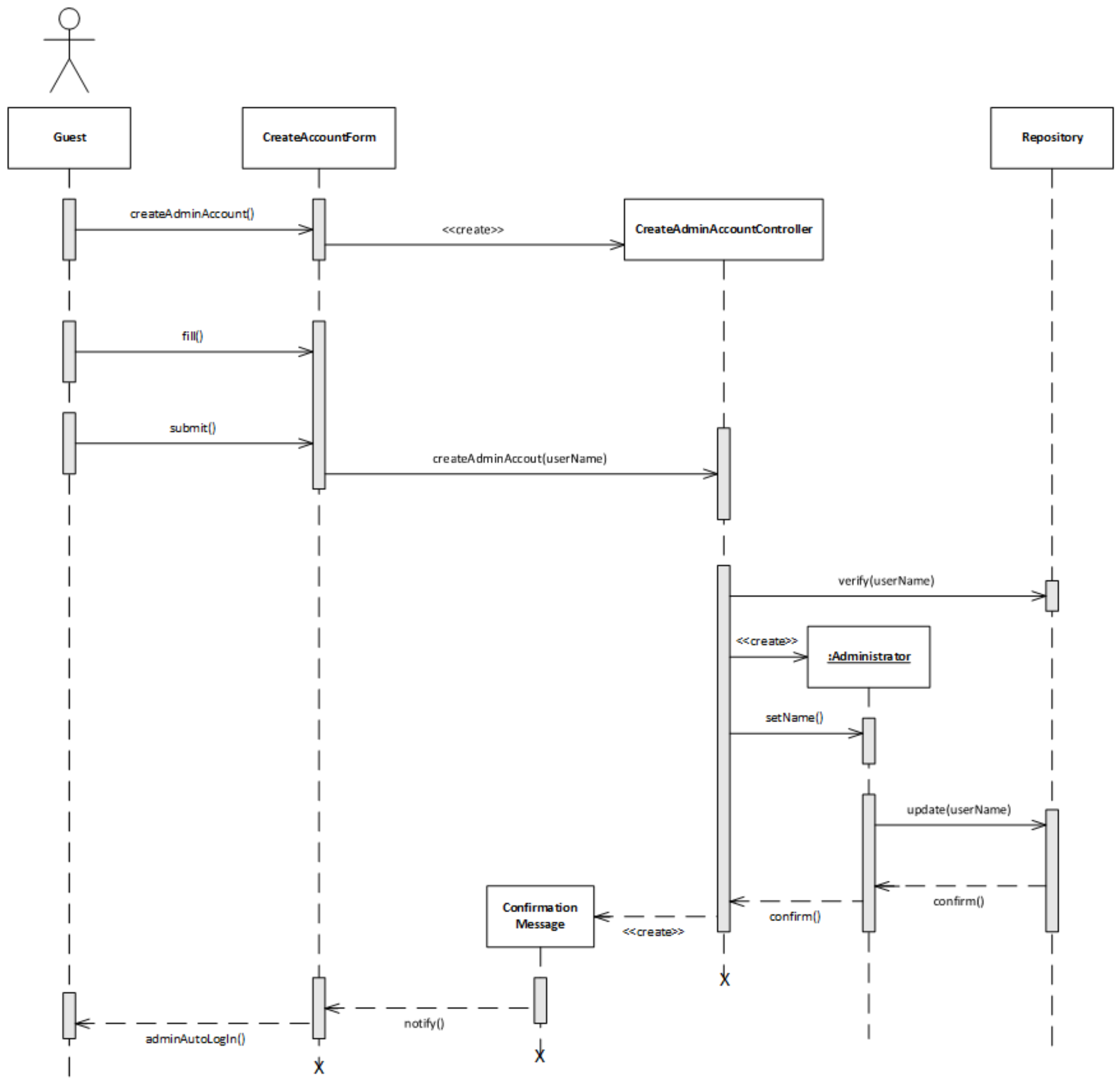

Diagram S-02: Sequence diagram for CreateAdminAccount Case

The CreateStudentAccount use case allows the Guest to create a new student account.

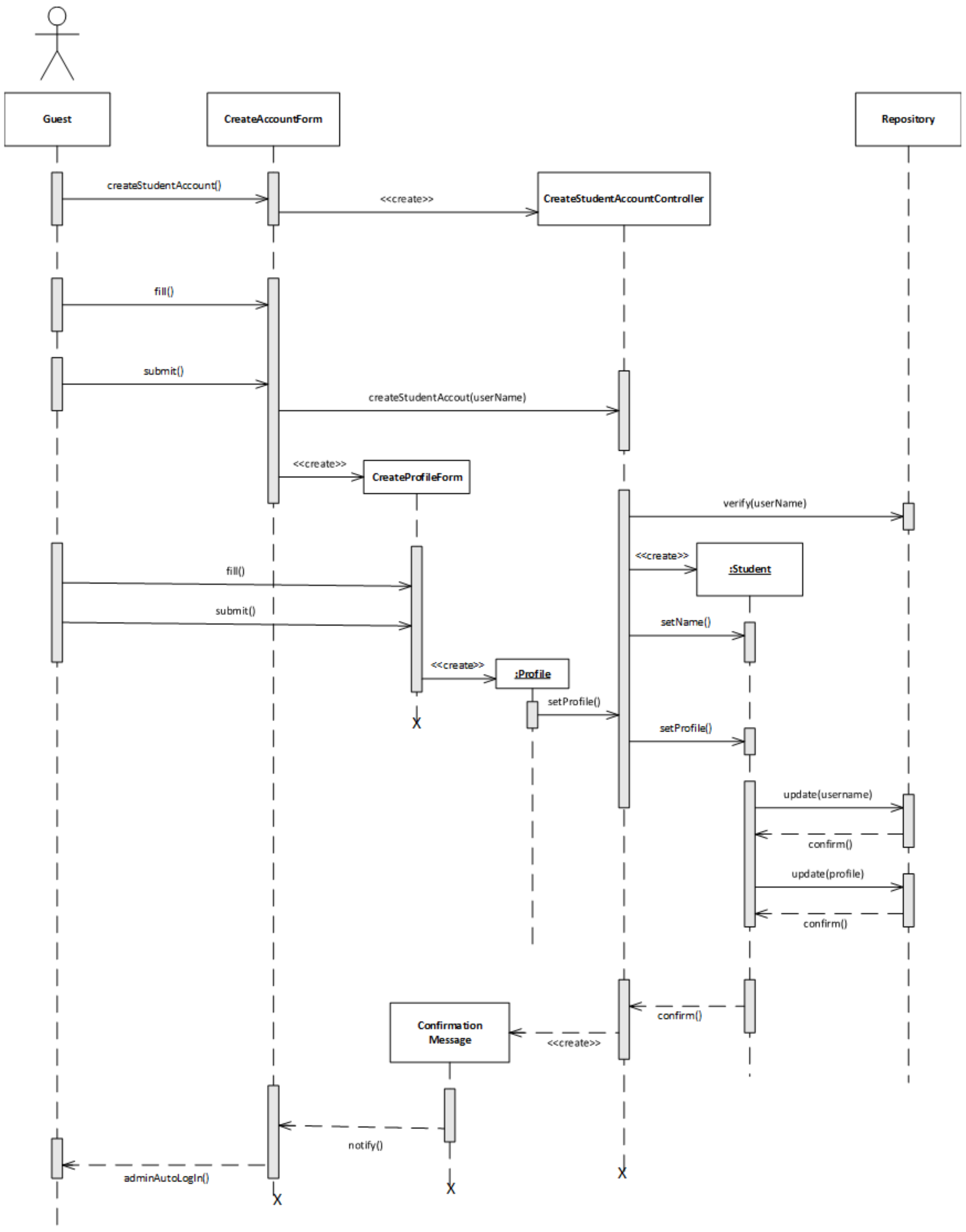

Diagram S-03: Sequence diagram for CreateStudentAccount Case

The EditProfile use case allows the Student to make changes to the qualifications and preferences in their own profile.

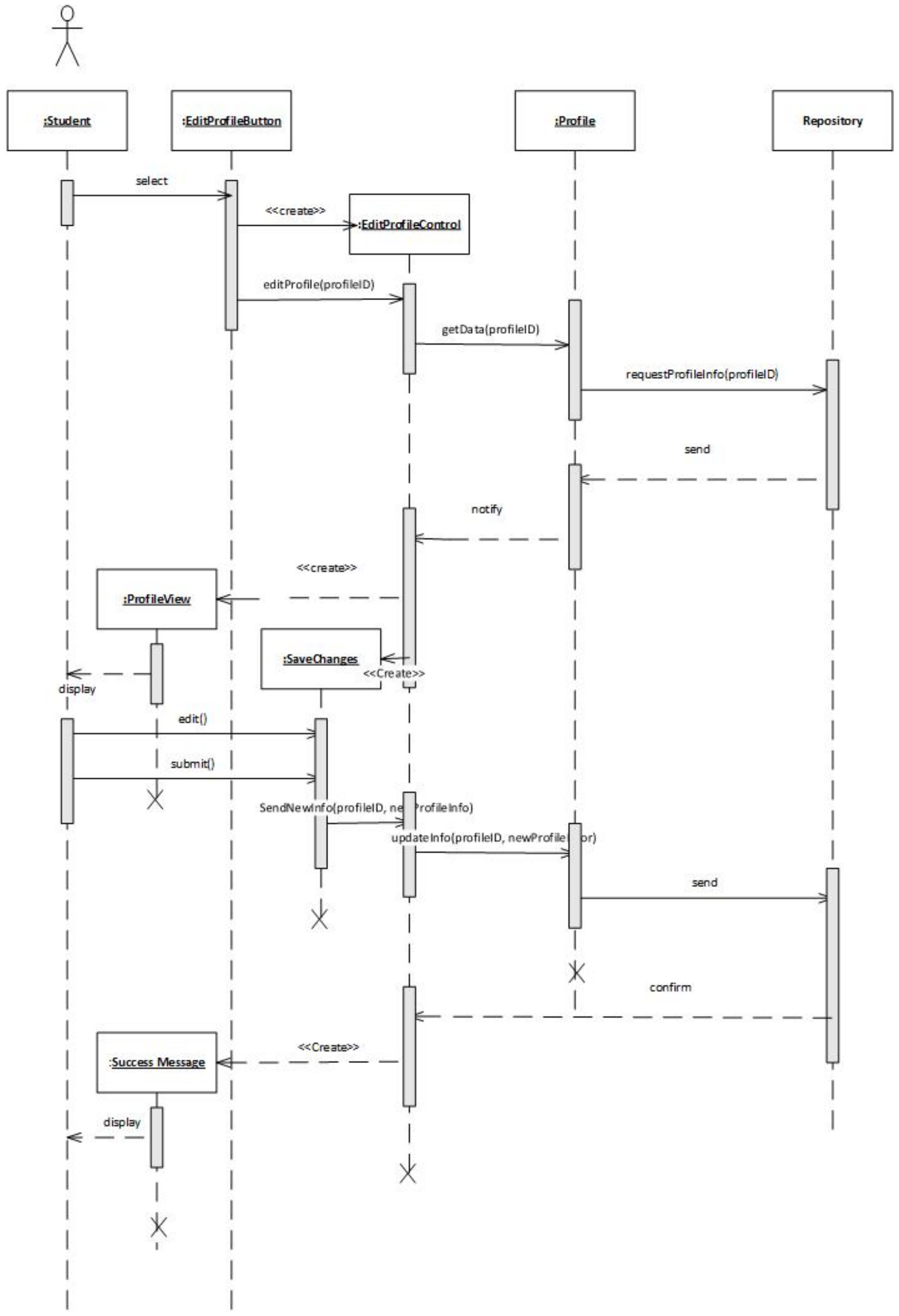

Diagram S-04: Sequence diagram for EditProfile Case

The JoinProject use case allows the Student to register with an existing project.

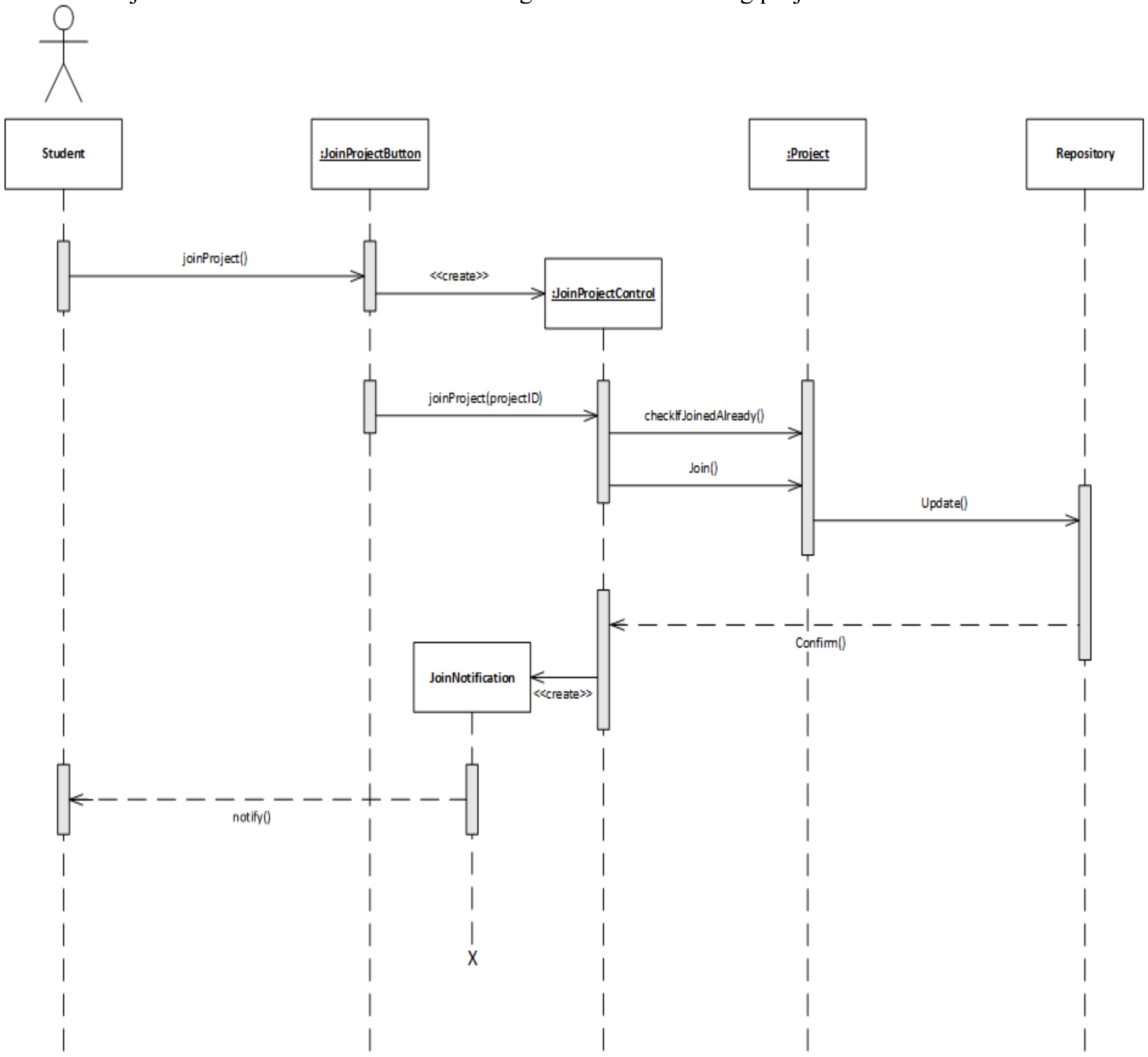

Diagram S-05: Sequence diagram for JoinProject Case

The ComputeBestTeams use case allows the administrator to find the optimal configuration of students to teams for a project and review the results in either summary or detailed form.

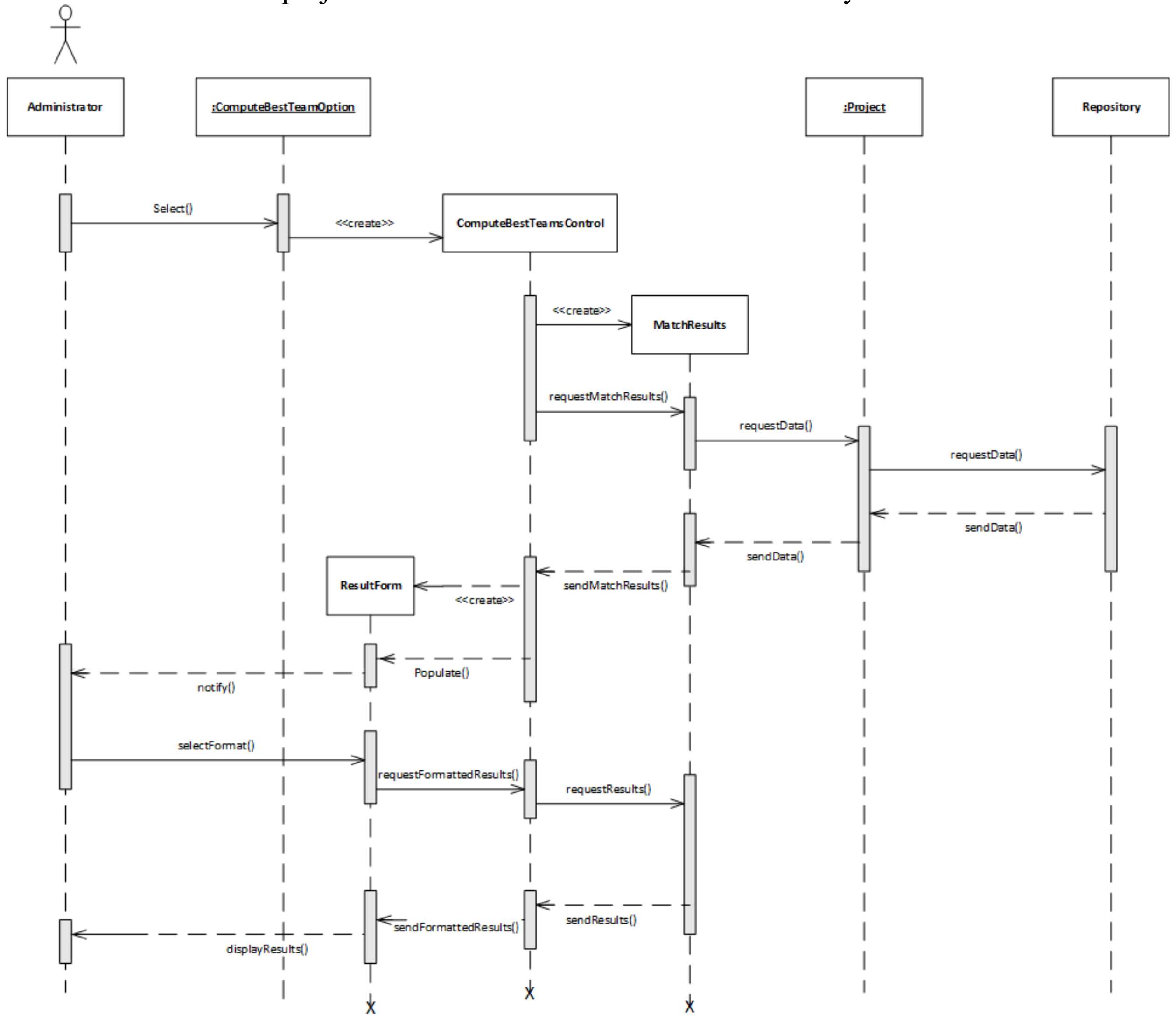

Diagram S-06: Sequence diagram for ComputeBestTeams Case

The CreateProject use case allows an Administrator to make a new project and define its properties such as title, description, and team size.

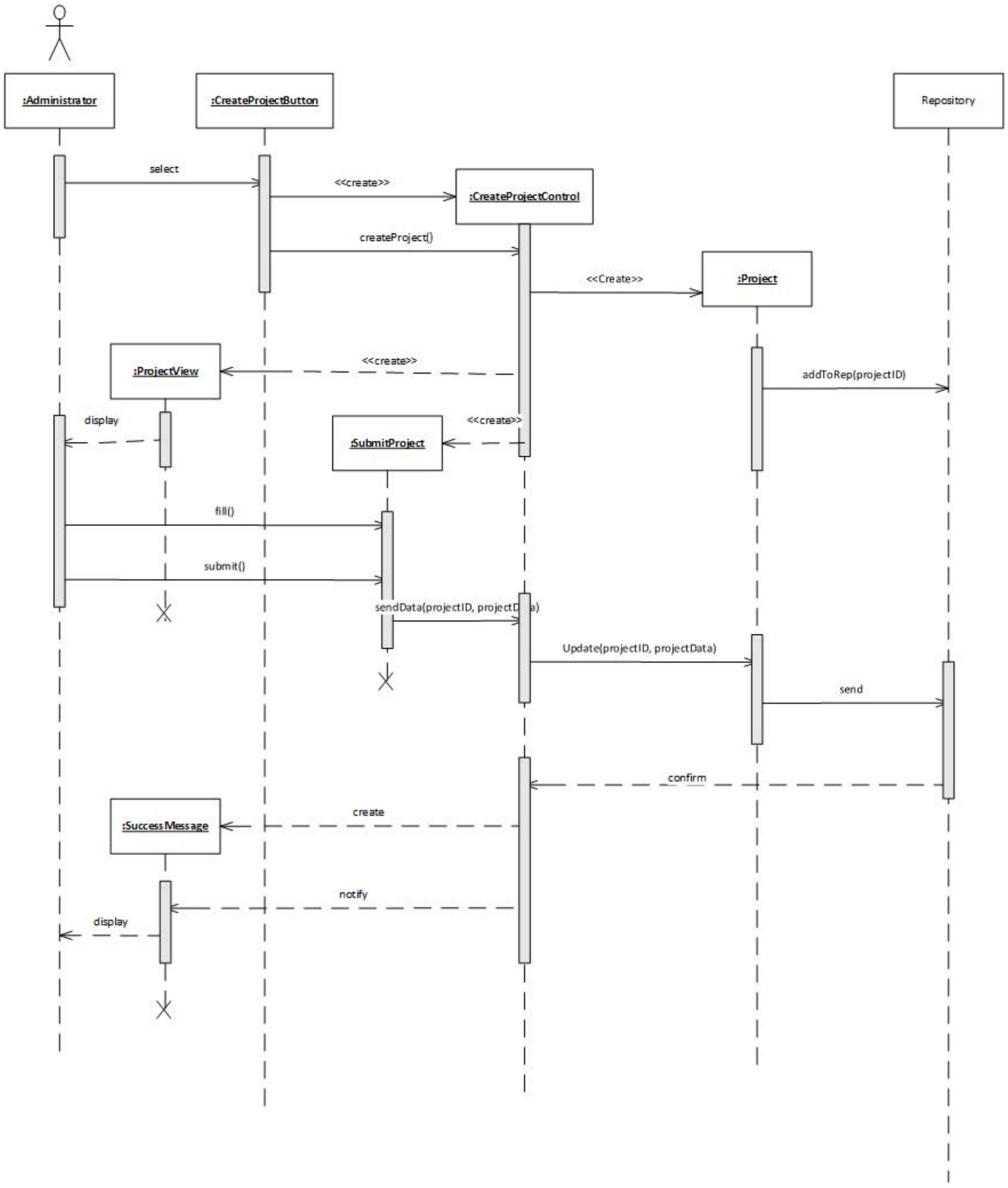

Diagram S-07: Sequence diagram for CreateProject Case

The EditProject use case allows an Administrator to modify the settings of an existing project, such as title, description, and team size.

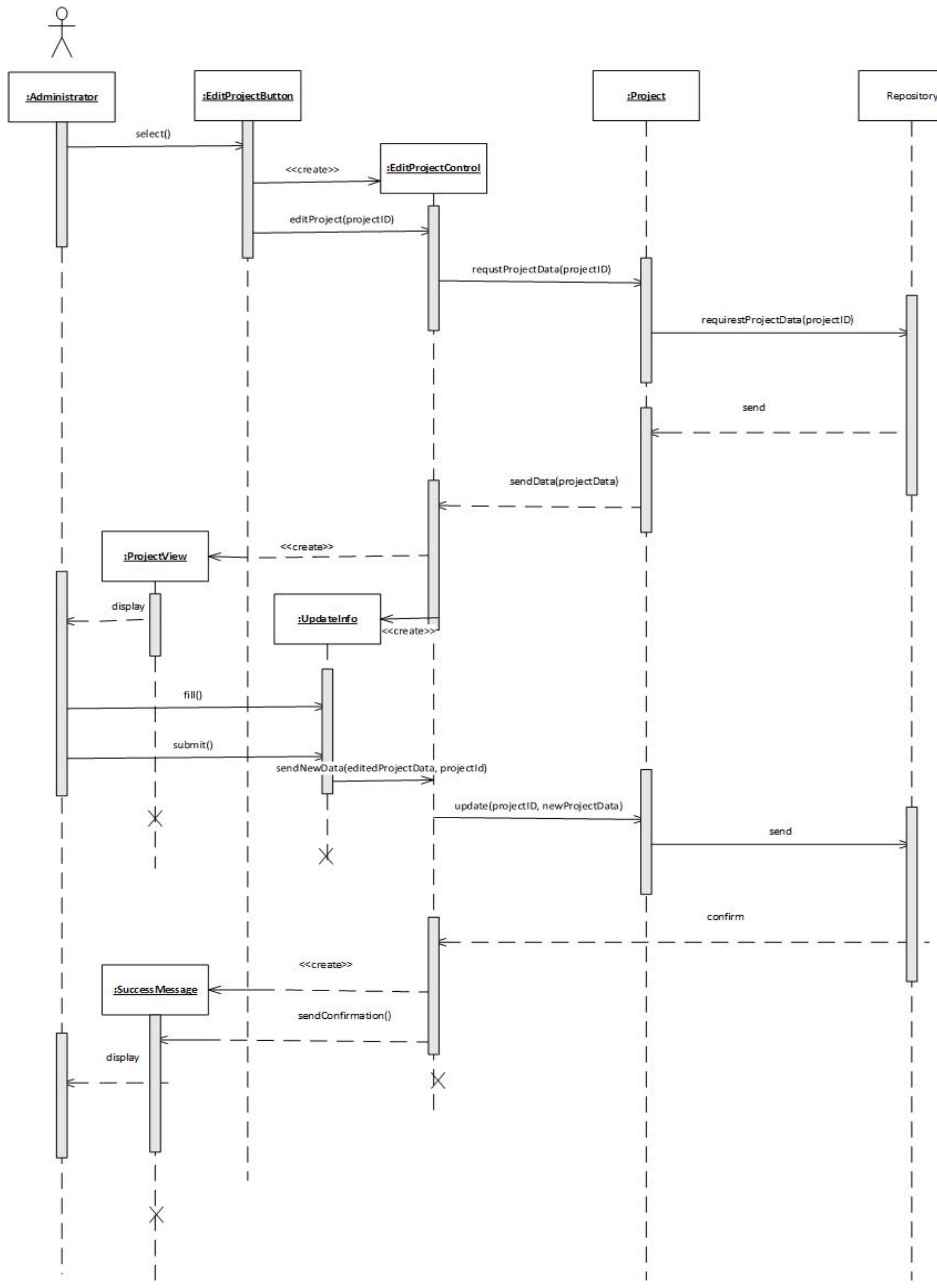

Diagram S-08: Sequence diagram for EditProject Use Case

The ViewExistingProjects use case allows a RegisteredUser (actually a Student or Administrator) to see a list of existing projects in the system.

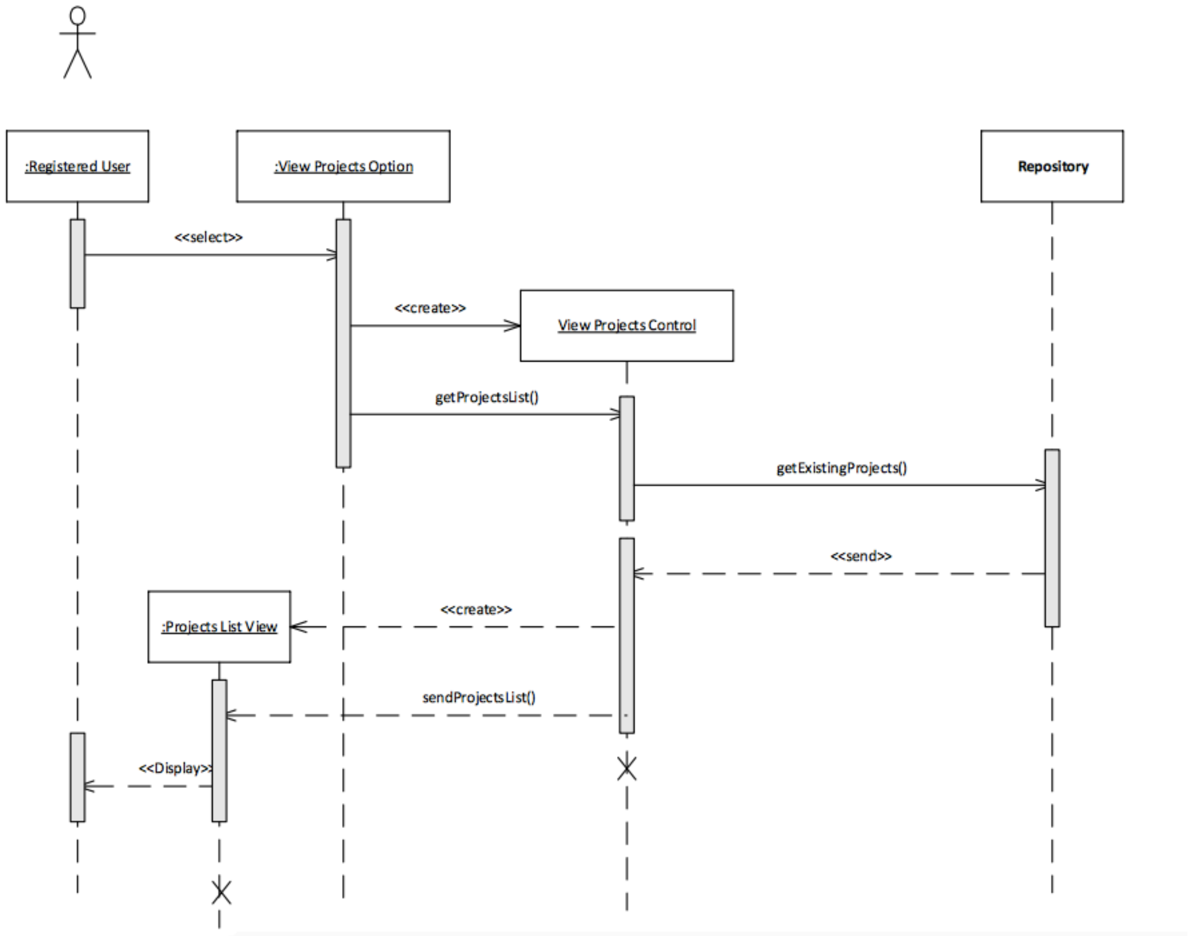

Diagram S-09: Sequence diagram for ViewExistingProjects

The AccountNameError use case allows the system to handle the error case in CreateAccount where the Guest provides an account name that is invalid or is already in use by another user account in the system.

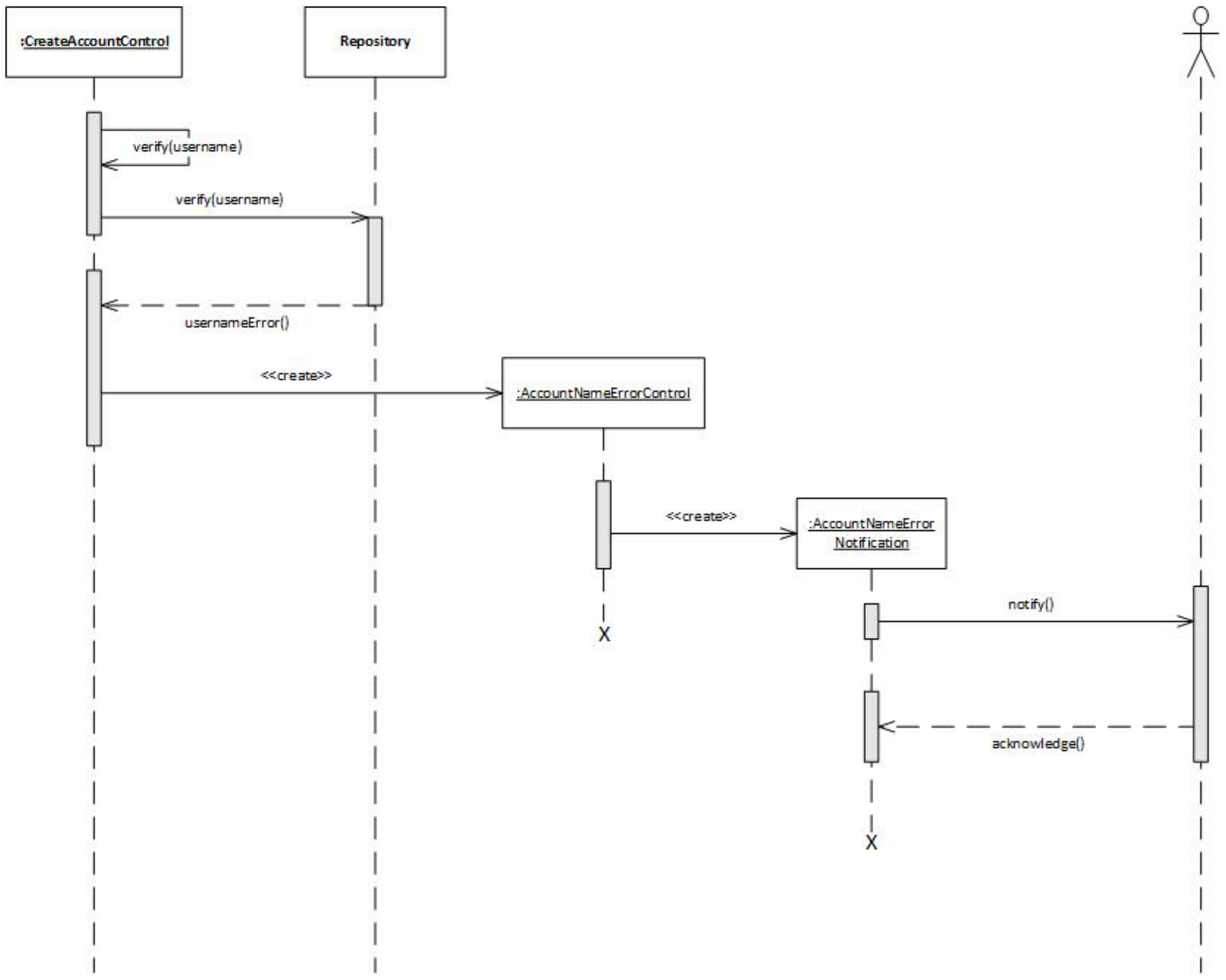

Diagram S-10: Sequence diagram for AccountNameError

The InvalidInputError use case allows the system to handle generic error cases where a Student, Administrator, or Guest provides invalid input in another use case.

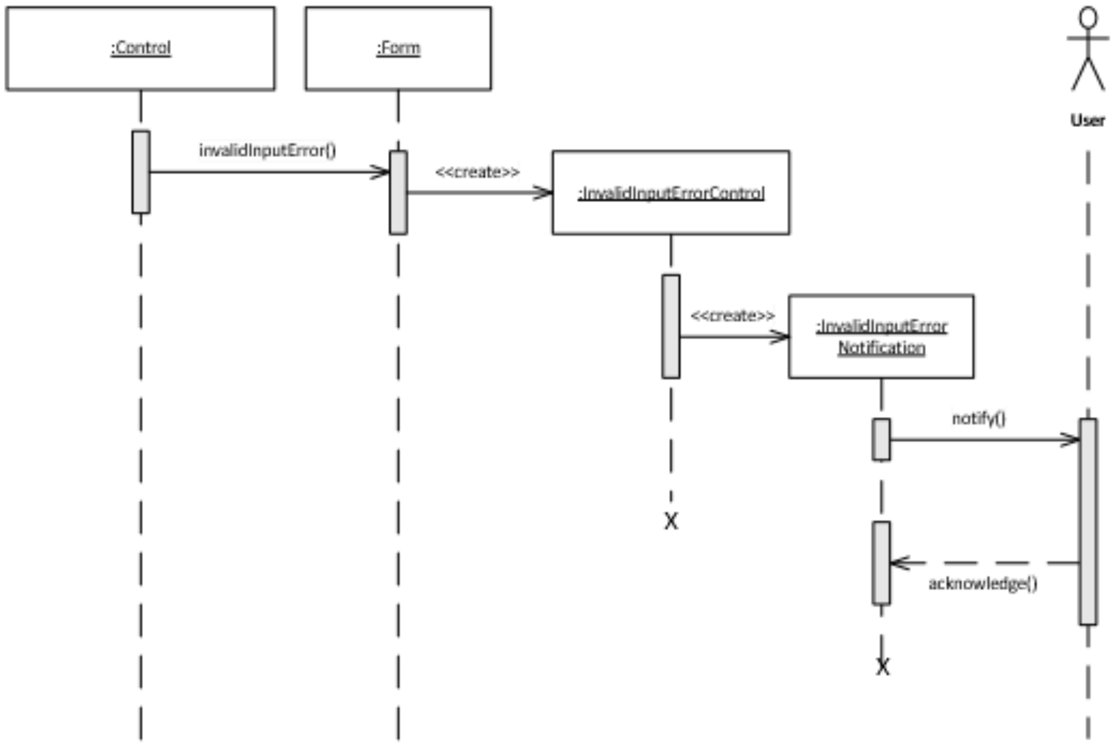

Diagram S-11: Sequence diagram for InvalidInputError

The NotEnoughStudentsError allows the system to handle the error case in ComputeBestTeams where the Administrator attempts to compute the teams for a project where there are no students registered or it is not possible to meet the minimum size requirement of all teams given the number of students and the project settings.

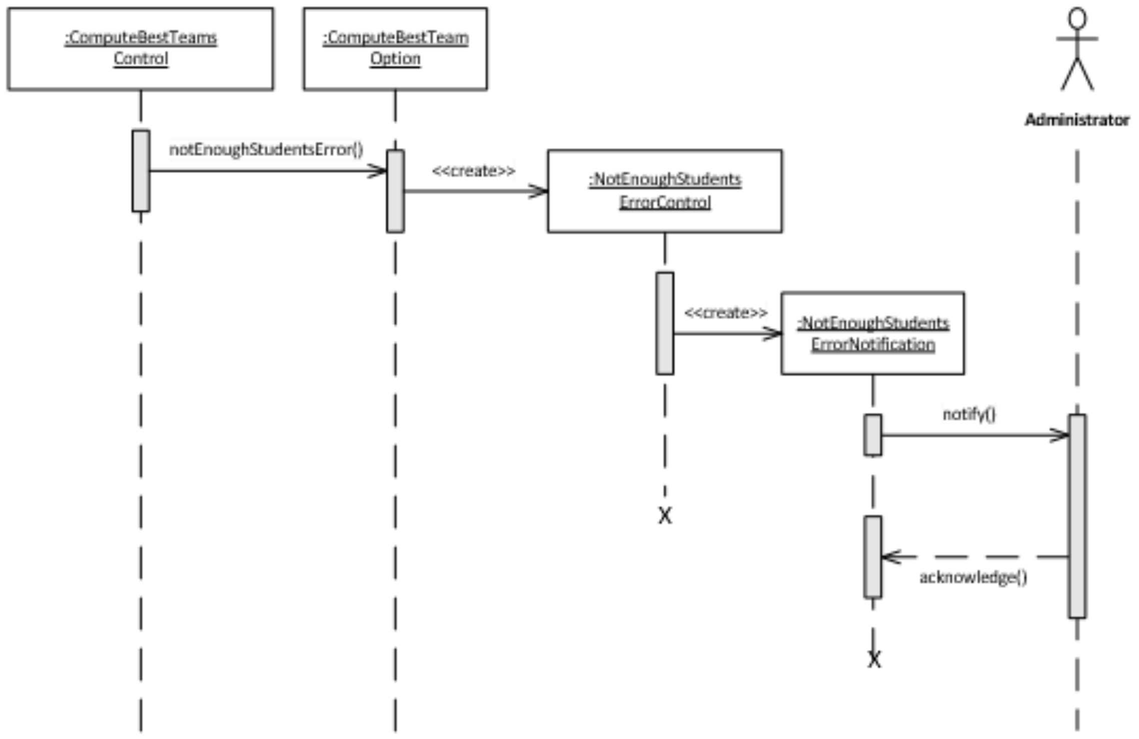

Diagram S-12: Sequence diagram for NotEnoughStudentsError

The ProfileError use case allows the system to handle the error case in CreateProfile where the Guest provides invalid information for the qualifications or preferences.

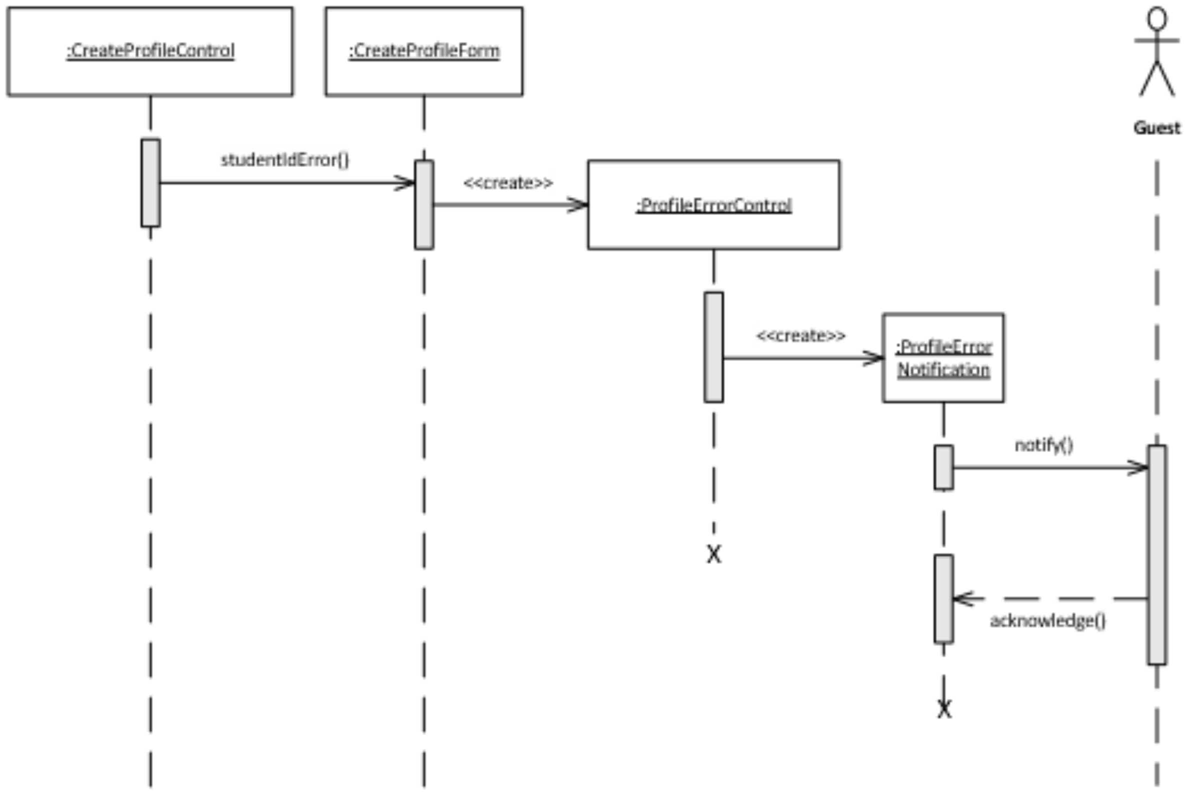

Diagram S-13: Sequence diagram for ProfileError

The RepositoryError use case allows the system to handle generic error cases stemming from issues related to the data repository, such as reading data, writing data, opening the repository, or data corruption.

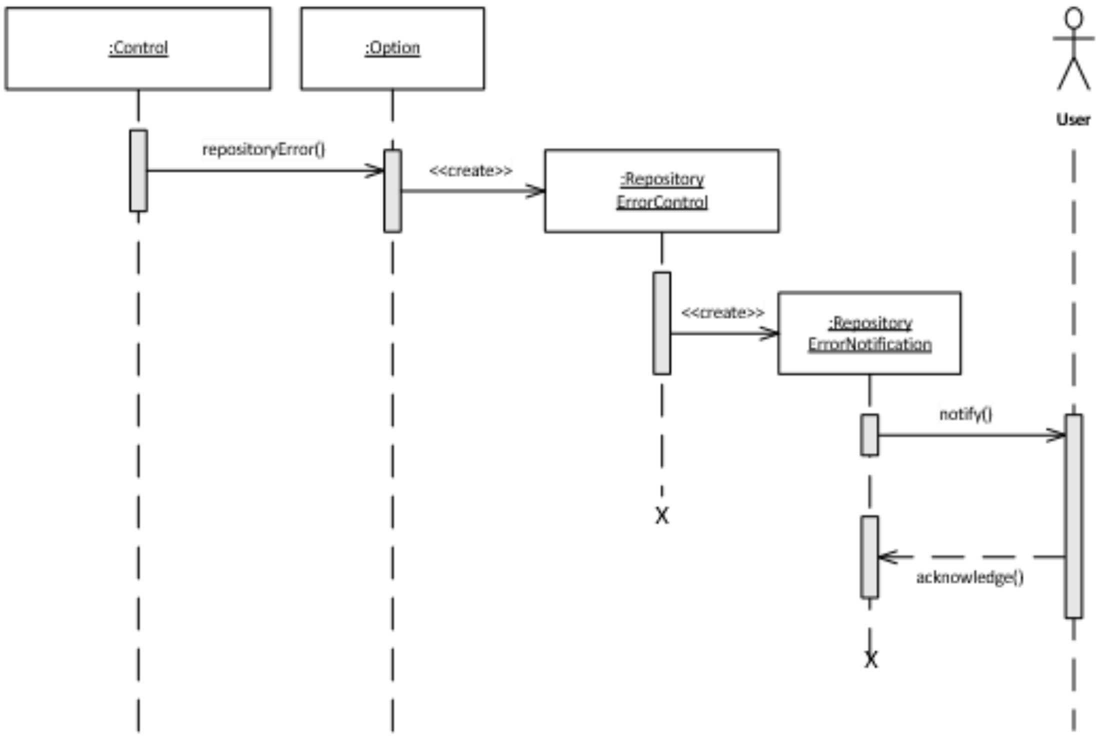

Diagram S-14: Sequence diagram for RepositoryError

The StudentIDError use case allows the system to handle the error case in CreateStudentAccount where the Guest provides a student ID that is invalid or already being used by another student account in the system.

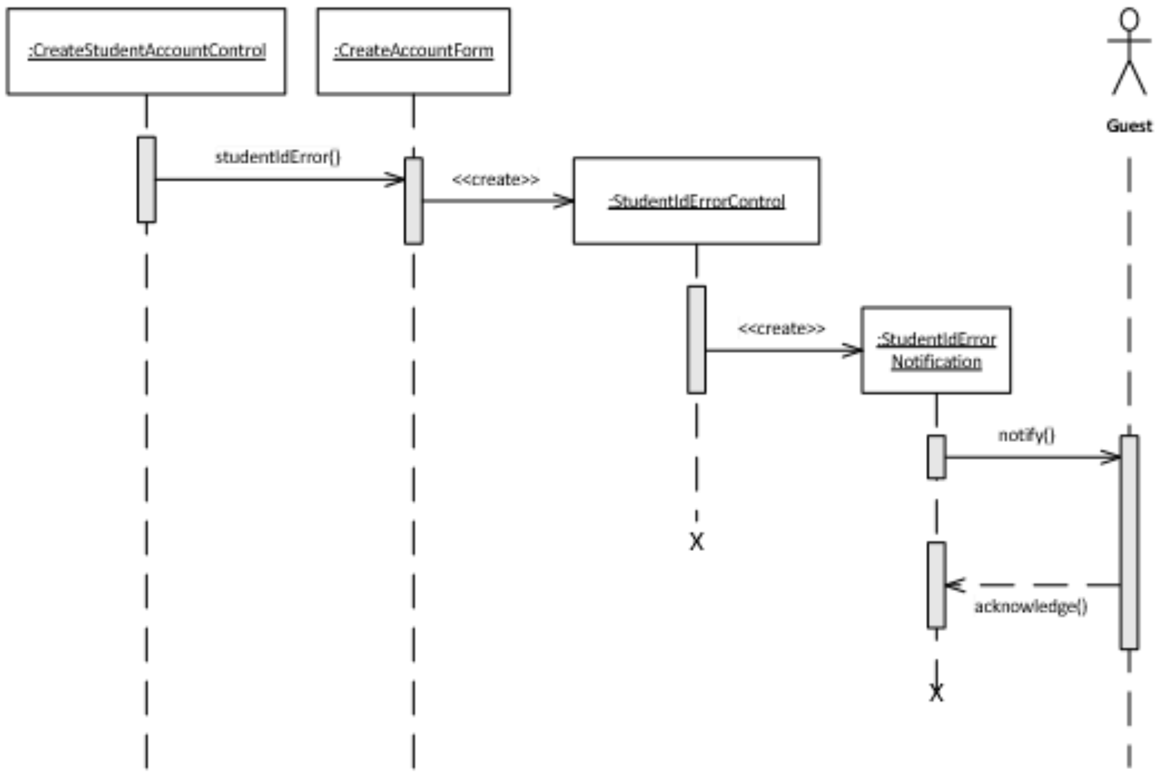

Diagram S-15: Sequence diagram for StudentIDError

3. Reference

3.1 Tables

3.2 Diagrams