Robotics Scoring Application

Software Requirements Specification

Version 2.3.1

October 1, 2015

Drew Erny
Jacob Reeves
Kelly Kashuda

Prepared for
CS 495—Capstone Senior Project
Instructor: Jeff Gray, Ph.D.
Fall 2015

Revision History

Date	Description	Author	Version
2015-09-18	Initial Draft	Drew Erny	1.0.0
2015-09-20	Added half of activity diagrams	Kelly Kashuda	1.1.0
2015-09-21	Added other half of activity diagrams	Jacob Reeves	1.2.0
2015-10-01	Made small tweaks from fault list	Drew Erny	1.2.1
2015-10-01	Added function requirements to activity diags.	Jacob Reeves	1.2.2
2015-10-01	Added two functional requirements	Drew Erny	1.3.0
2015-10-01	Added definition of team captain	Kelly Kashuda	1.3.1
2015-10-01	Put diagrams on their own pages.	Drew Erny	1.3.2
2015-11-02	Fixed numbering mistake in Func. Req.	Drew Erny	1.3.3
2015-11-02	Added first half of Sequence Diagrams	Jacob Reeves	2.0.0
2015-11-02	Added second half of Sequence Diagrams	Drew Erny	2.1.0
2015-11-02	Added Class Diagrams	Kelly Kashuda	2.2.0
2015-11-02	Added Routing Table	Drew Erny	2.3.0
2015-11-02	Fixed some typos	Jacob Reeves	2.3.1

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for the Robotics scoring application. This document provides a high-level overview of the functionality of the application, functional and non-functional requirements, and diagrams describing core functionality and architecture. This document should provide adequate instruction for a development team to begin work on the application.

1.2 Scope

The Robotics Scoring Application will provide a robust and easy to use replacement for the current paper-to-spreadsheet scoring system. It will keep track of teams' scores, determine competition winners, store data from previous years, and allow the export of data to CSV format. It will provide robust administration access, so that parameters can be tweaked for the specifics of each year's competition. It will be made to not lose data and provide adequate feedback so that judges and teams can know that data was correctly entered.

1.3 Definitions, Acronyms, and Abbreviations

Administrator:	A master user with the power to view and edit global scoring data
Attempt:	One play of a Challenge, which yields a score. Teams may attempt challenges multiple times.
Challenge:	A game type. Teams are scored based on how well they meet the goals of the challenges.
Course:	A particular physical location or playing field where the Challenge is attempted. There may be more than one per challenge
Judge:	A person who scores a challenge and enters the Attempt data into the system.
Team:	A group fielded by a school. There may be more than one team per school.
Team Captain:	A person responsible for approving a team's score for an attempt.

2. Project Description

2.1 Product Perspective

Every year, Dr. Jeff Gray and the University of Alabama Computer Science department host a robotics competition open to elementary, middle, and high school students. As many as 60 teams attend and compete in three different challenges. In the past, scoring this competition has relied on judges recording scores on paper, which are then transported to a central location to be entered en masse into a spreadsheet, which is then analyzed to determine winners. This system is

inefficient and tedious, and a relatively simple web-based solution can considerably improve the scoring process.

2.2 Product Functions

2.2.1 Scoring

The system will provide an interface for judges to enter attempt data. The judges will access the scoring system through a web browser on either a mobile device or traditional computer, and log in with provided credentials. Each course will have a device for the judges. Upon completion of an attempt, the judges will enter scores and allow the team to verify that the data is correctly entered.

2.2.2 Determine Winners

The system will process scoring data so that winners may be determined. The administrator will be able to request a list of the teams, sorted by score, and grouped by grade level. The system will take into account the various metrics teams' overall rankings are computed from.

2.2.3 Export Data

The system will allow the administrator to export attempt data to a CSV file, so that the data can be examined if desired.

2.2.4 Administration Panel

The system will have an administration panel, with which the administrator can enter the specifics of each years competition, including team data and course data. The administration panel will also allow for the issuance of login credentials for judges and provide a location from which to download the CSV export. The administration panel will be protected with a password.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Accept Scoring Data

3.1.1.1 Introduction

The application will accept data about a team's attempt from the judge.

3.2.1.2 Inputs

Team Name	The name of the team being scored.
Score	The numeric score received for the attempt.
Attempt Duration	The amount of time required to complete the attempt. May be blank for some challenges.
Completion Time	The time at which the attempt was completed.
Judge Name	The name of the judge scoring the attempt.

3.1.1.3 Processing

The inputs are saved to a database.

3.1.1.4 Outputs
The data is displayed back so the user can see that it has been successfully saved.

3.1.1.5 Error Handling
If any fields are not valid, the judge must repeat the data entry, modifying the incorrect fields. If data is not successfully saved, the user is notified.

3.1.2 Determine Winners

3.1.2.1 Introduction
The application will determine the winning teams.

3.1.2.2 Inputs
The attempt data is read from the database. The user specifies a grade level to be selected.

3.1.2.3 Processing
The attempt data is sorted, first by highest attempt score by challenge, then by shortest attempt duration, then by earliest completion time.

3.1.2.4 Outputs
A list of teams is returned, sorted by highest attempt ranking computed in the previous step.

3.1.2.5 Error Handling
If an invalid grade level is supplied, the application asks the administrator to select a valid grade level.

3.1.3 Export Data

3.1.3.1 Introduction
The software will allow the administrator to export data for attempts in previous years.

3.1.3.2 Inputs
The administrator input the year for which the attempt data is requested.

3.1.3.3 Processing
The attempt data for the requested year is read from the database.

3.1.3.4 Outputs
A CSV-formatted document is returned with de-normalized attempt data, including columns with full team data.

3.1.2.5 Error Handling
If an invalid year is requested, an empty CSV will be returned. If an error in processing occurs, the administrator will be notified.

3.1.4 Configure Year's Game Data

3.1.4.1 Introduction
The administrator needs to input the game data for the year's game. Different years may have different teams competing, different challenge parameters, different numbers of courses, etc.

3.1.4.2 Inputs
Team information, challenge information (whether or not a particular challenge includes attempt duration as a scoring metric), course information (how many courses and what challenges they are associated with).

3.1.4.3 Processing
The data is stored in a database so that the year's competition can be accurately modeled. Attempt data will reference this configuration.

3.1.4.4 Outputs
No outputs.

3.1.4.5 Error Handling
If invalid data is input, the administrator will be notified and given a chance to correct the data. If an error in processing occurs, the administrator will be notified.

3.1.5 View Team Scores

3.1.5.1 Introduction
The administrator or the judges should be able to view a team's scores at any time during the game.

3.1.5.2 Inputs
Team name.

3.1.5.3 Processing
The team's attempt data is all retrieved. The top scores in each category are marked, and the overall score as a sum of the scores is generated

3.1.5.4 Outputs
The team's attempt data, with the top scores featured prominently, is displayed, as well as a sum.

3.1.5.5 Error Handling
If an invalid team name is given, the user can enter a new team name. If an error in processing occurs, the user will be notified.

3.1.6 Allow Team Captain to Approve Score

3.1.6.1 Introduction
The application should allow a team captain to verify that scoring data has been entered correctly, and sign off if it has.

3.1.6.2 Inputs
The software will display to the team captain data of the attempt as it has been entered by the judge. The team captain will be able to accept or reject the judge's score

3.1.6.3 Processing
If the team captain selects approve, the software will commit the attempt to the database. If the team captain rejects, the attempt is not committed and the judge will be able to amend it.

3.1.6.4 Outputs
A verification that the attempt data has been successfully committed will be displayed, or the original attempt data in a form for editing will be displayed if the attempt is rejected.

3.1.6.5 Error Handling
If an error in processing occurs, the administrator will be notified.

3.1.7 Login

3.1.7.1 Introduction
The administrator and judges will be able to log in.

3.1.7.2 Inputs
The course name or administrator name and the password

3.1.7.3 Processing
The username and password are checked against the database

3.1.7.4 Output
The user is given an authenticated session.

3.1.7.5 Error Handling
If the username or password is incorrect, the user is alerted.

3.1.8 Retrieve Course Credentials

3.1.8.1 Introduction
The administrator will be able to retrieve the passwords for each course.

3.1.8.2 Inputs
The course to retrieve credentials for

3.1.8.3 Processing
None

3.1.8.4 Outputs
The password for the course.

3.1.8.5 Error Handling
If an error occurs, the administrator will be notified.

3.2 Use Cases

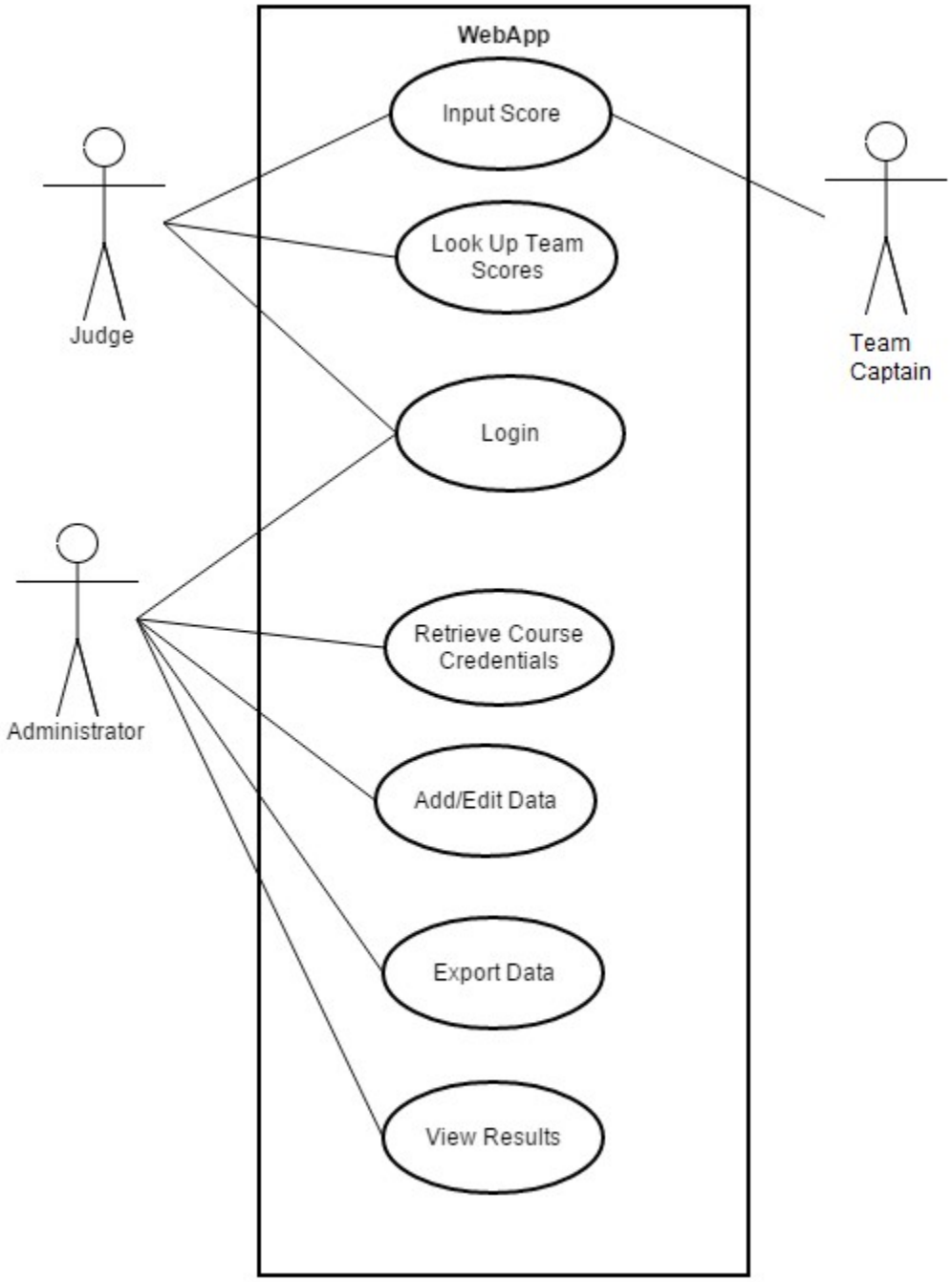


Figure 1: A use case diagram outlining all of the use cases of the software

3.2.1 Input Score

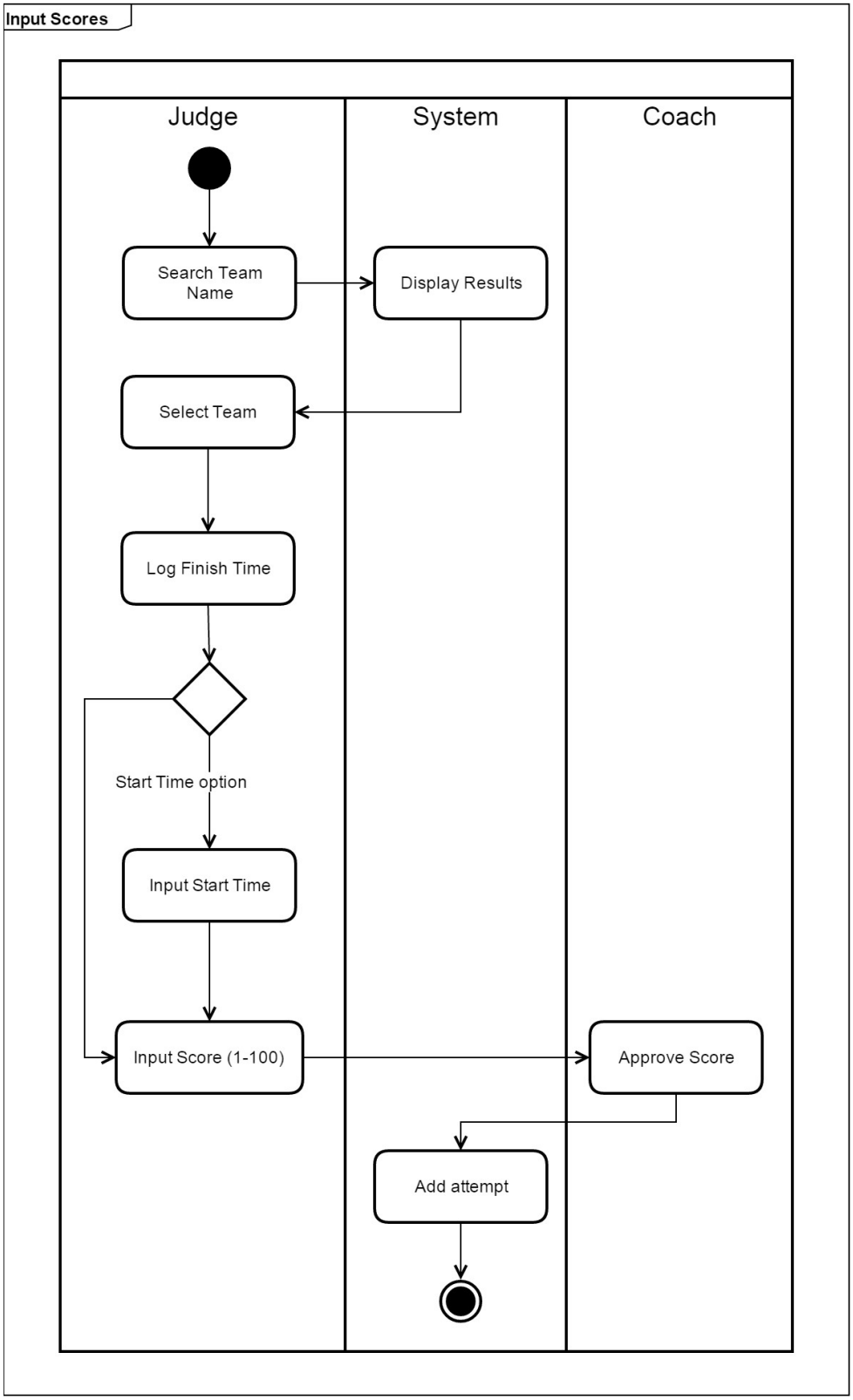


Figure 2: Activity diagram for the "Input Score" use case

3.2.1.1 Description

Figure 2 describes the steps to input scores. The judge first searches the team name, and the system responds by returning a result if the team name exists. Searching provides autocomplete. The judge then selects the team and is prompted with a view to input the score information. If the start time is important to the specific course, the judge enters it, otherwise the judge enters only the end time. The judge then inputs the numeric score (1-100), and finally the team captain of the specific team approves the score.

3.2.1.2 Functional Requirements Fulfilled

Figure 2 describes the process to fulfill functional requirement 3.1.1.

3.2.2 Look Up Team Scores

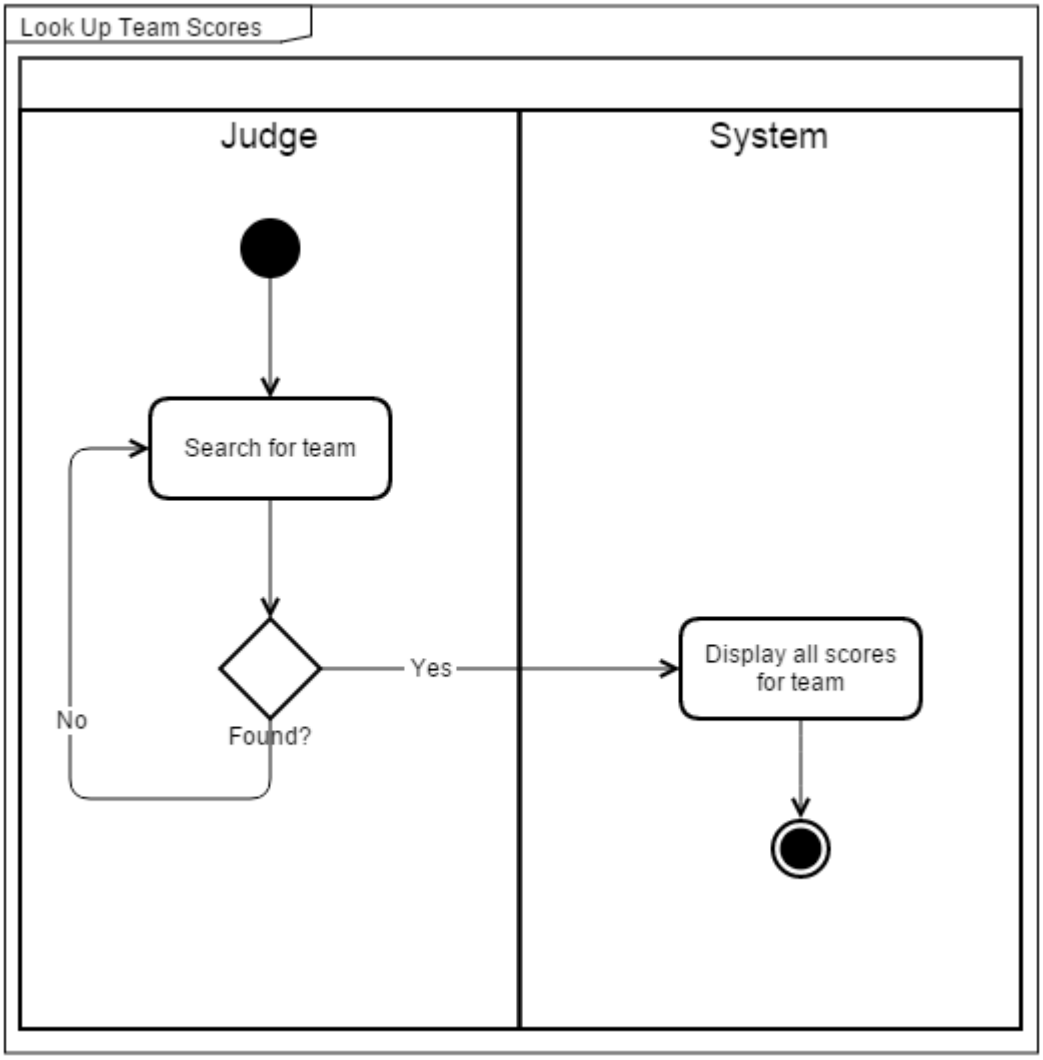


Figure 3: Activity diagram for the "Look Up Team Scores" use case.

3.2.2.1 Description
Figure 3 describes how a user is able to look up scores for a specific team. The judge will search for a team name, once it is found the judge can select it and the scores will be displayed for that specific team. Searching provides autocomplete.

3.2.2.2 Functional Requirement Fulfilled
Figure 3 describes how to fulfill functional requirement 3.1.5.

3.2.3 Login

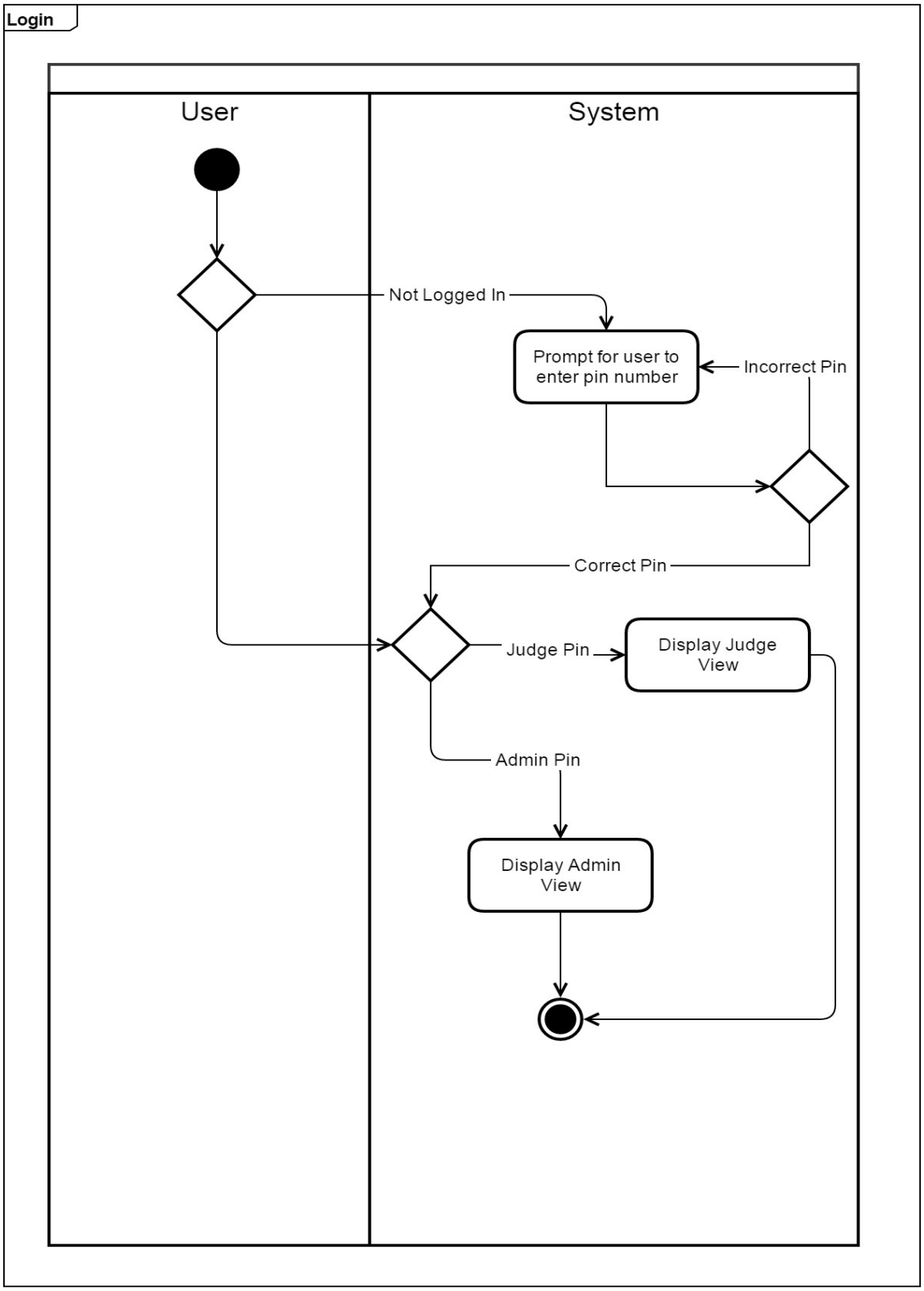


Figure 4: Activity diagram for the "Login" use case.

3.2.3.1 Description

Figure 4 describes the process of logging into the web app. If no one is currently logged in, the system will prompt the user with a screen asking the user to enter a 4-digit pin. As long as the pin entered is incorrect, the system will continue prompting the user.

3.2.3.2 Functional Requirement Fulfilled

Figure 4 describes how to fulfill functional requirement 3.1.6.

3.2.4 Retrieve Course Credentials

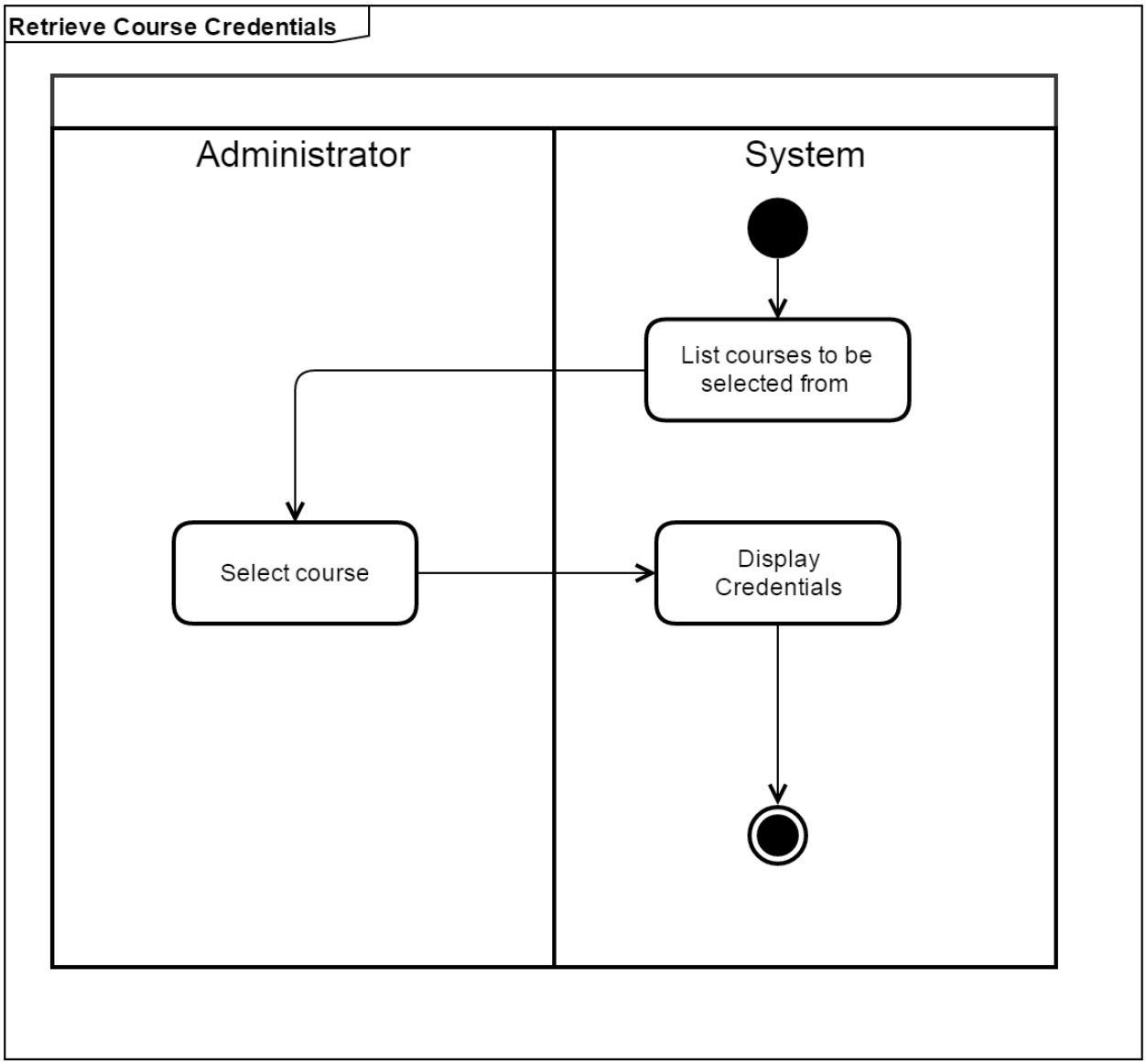


Figure 5: Activity diagram for the "Retrieve Course Credentials" use case.

3.2.4.1 Description
Figure 5 describes the process of selecting credentials for a specific course. Once the user selects the menu option to retrieve the course credentials the system will then list the courses to be selected from. The user can then choose to view the credentials of a specific course. The system will display the credentials based on the course the user has chosen.

3.1.4.1 Functional Requirement Fulfilled
Figure 5 describes how to fulfill functional requirement 3.1.7.

3.2.5 Add/Edit Data

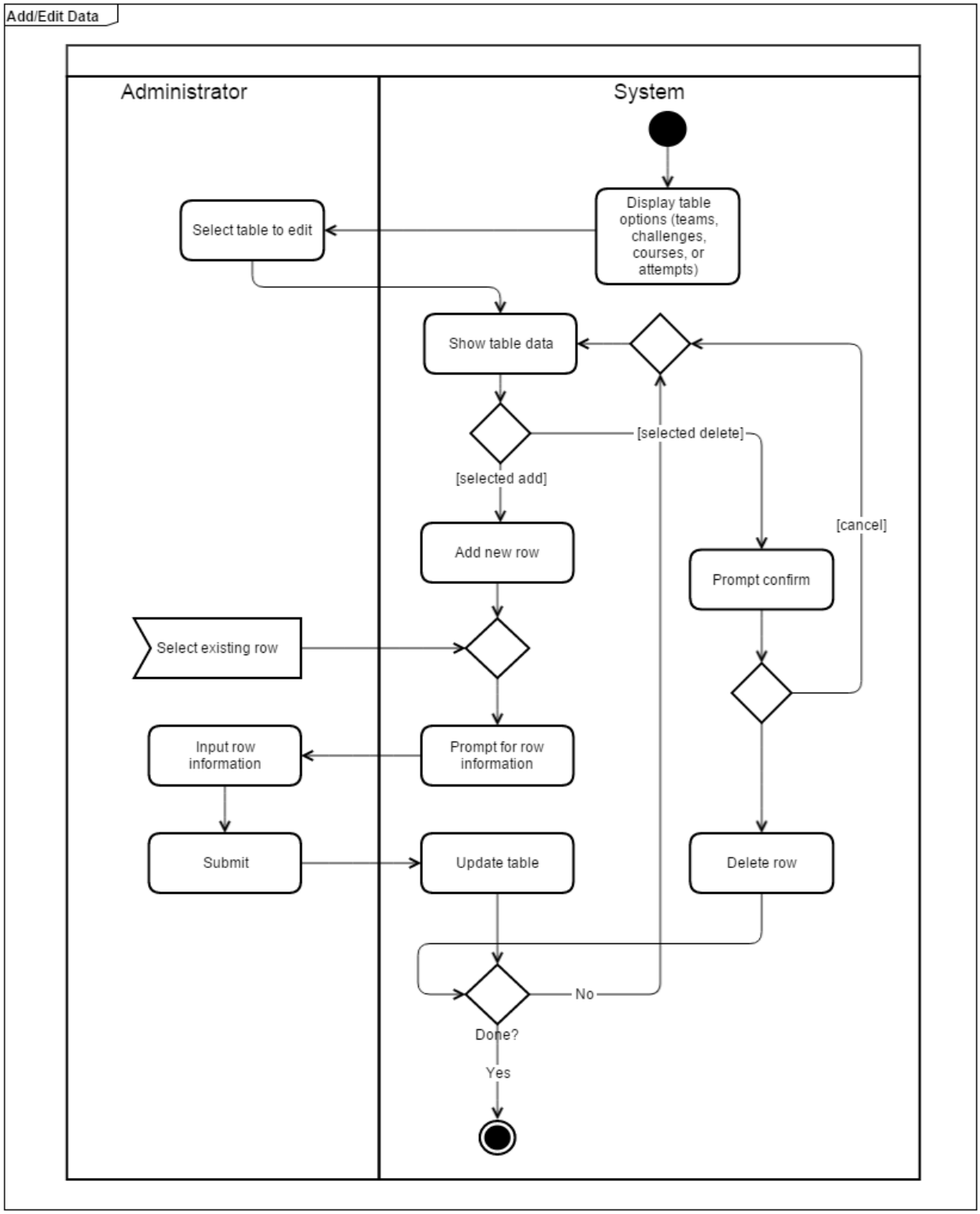


Figure 6: Activity diagram for the "Add/Edit Data" use case.

3.2.5.1 Description
Figure 6 describes how the administrator can add, edit, and delete data from the team, challenge, course, and attempt tables. The system displays the tables the administrator can edit. The administrator selects a table to edit. The system displays the current data for the selected table. At any time, the administrator may select an existing row to be prompted for revised information. The administrator may select either add or delete. If add was selected, the administrator is prompted for information to fill the new row. The administrator inputs the information and presses submit. The system updates the table with the new row. If delete was selected, the system prompts for confirmation. Once confirmed, the row is deleted. The administrator may continue adding, editing, or deleting rows if necessary.

3.2.5.2 Functional Requirement Fulfilled
Figure 6 describes how to fulfill functional requirement 3.1.4.

3.2.6 Export Data

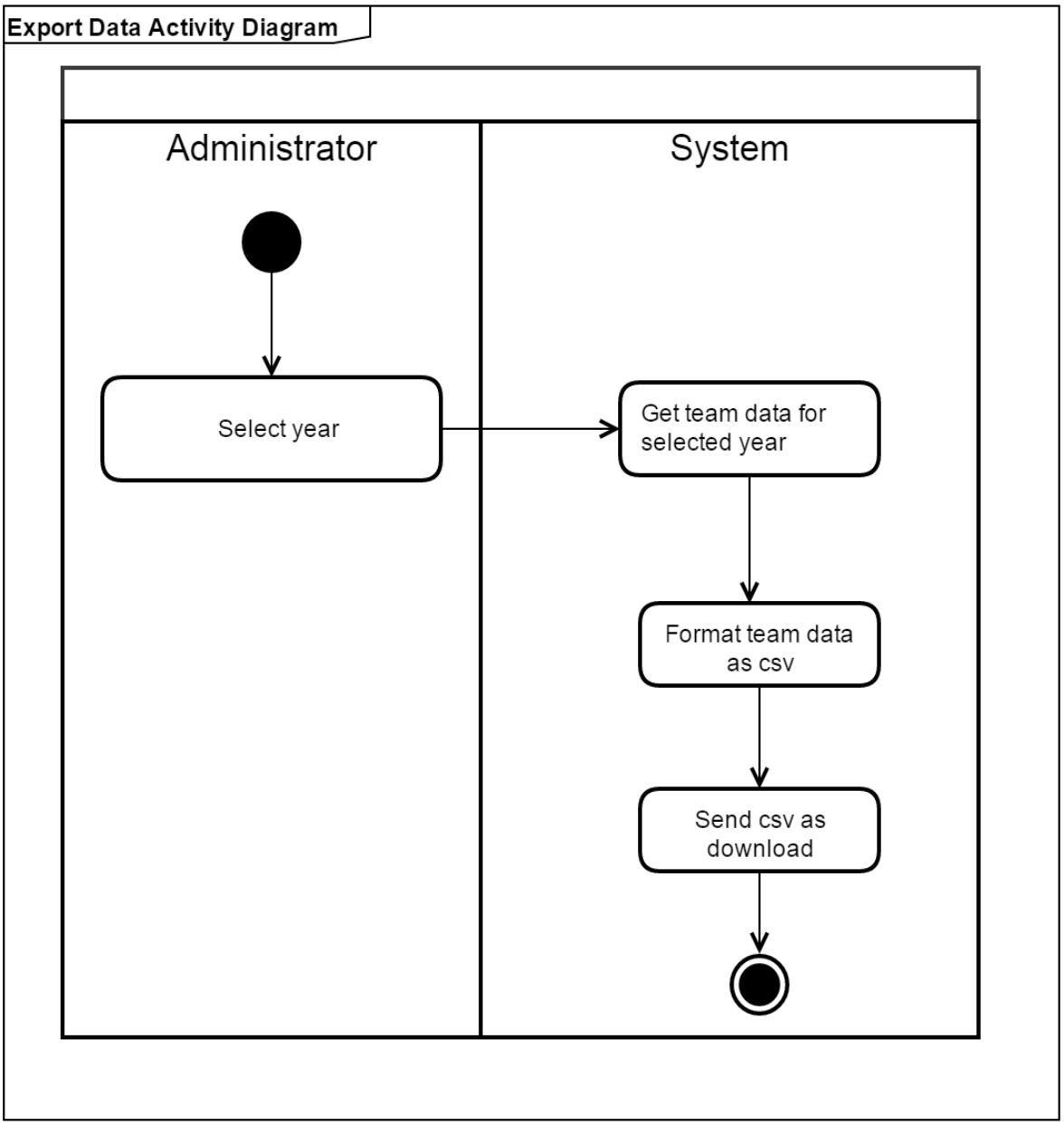


Figure 7: Activity diagram for the "Export Data" use case.

3.2.6.1 Description
Figure 7 describes how the administrator exports data from the system to a savable format. The administrator selects a year of data to export. The system gets the selected year's data and formats the data into a csv file. The system sends the csv file as a download.

3.2.6.2
Figure 7 describes how to fulfill functional requirement 3.1.3.

3.2.7 View Results

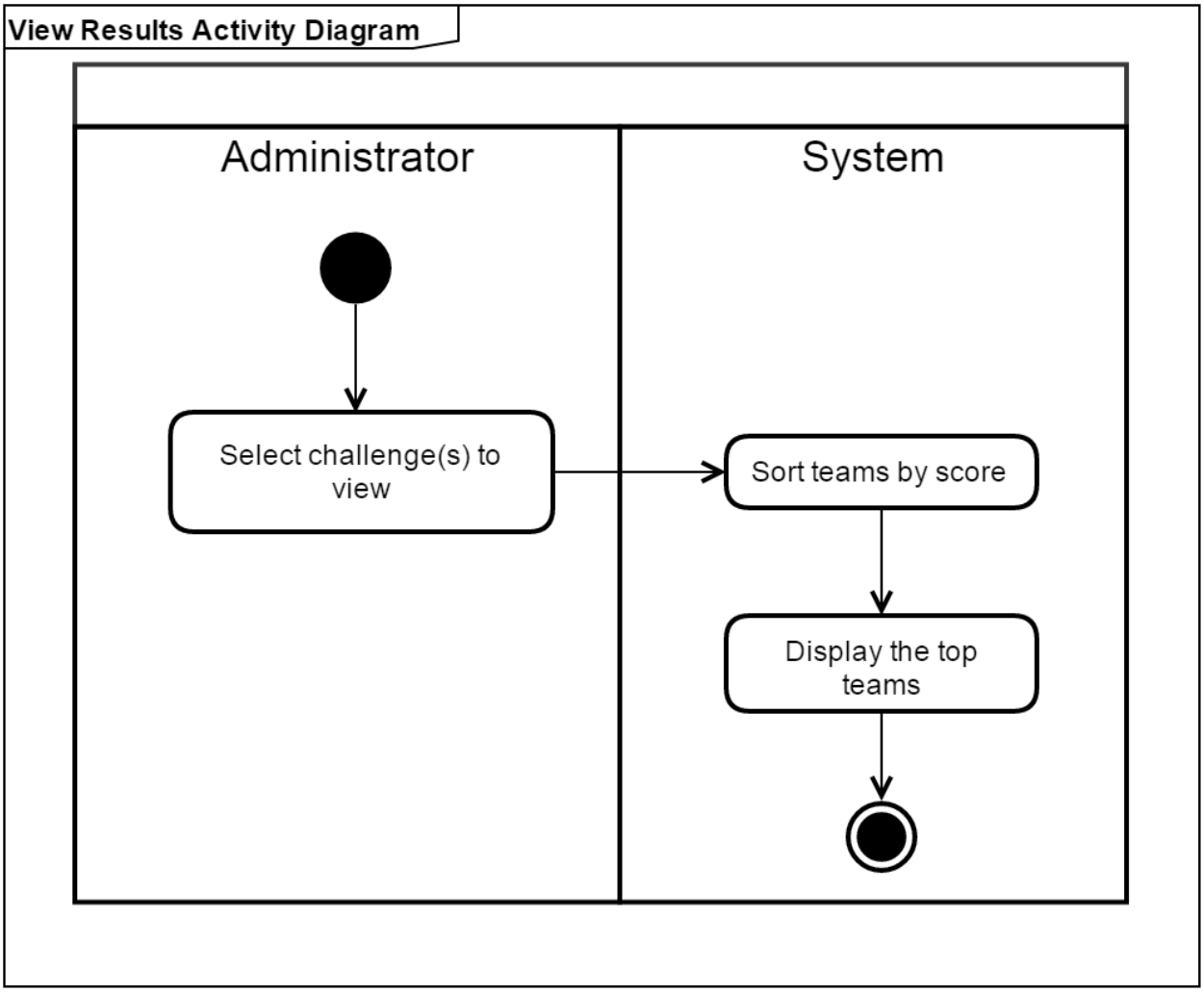


Figure 8: Activity diagram for the "View Results" use case.

3.2.7.1 Description

Figure 8 describes how the administrator may ask the system to display the winning teams. The administrator selects which challenge and age group, or the administrator may select overall winner. The system sorts the relevant teams by score, determining ties by the completion time of the challenge, and further by duration taken to complete the challenge if that is valid for the challenge. The system displays the sorted list of teams.

3.2.7.2 Functional Requirement Fulfilled

Figure 8 describes how to fulfill functional requirement 3.1.2.

3.3 Classes / Objects

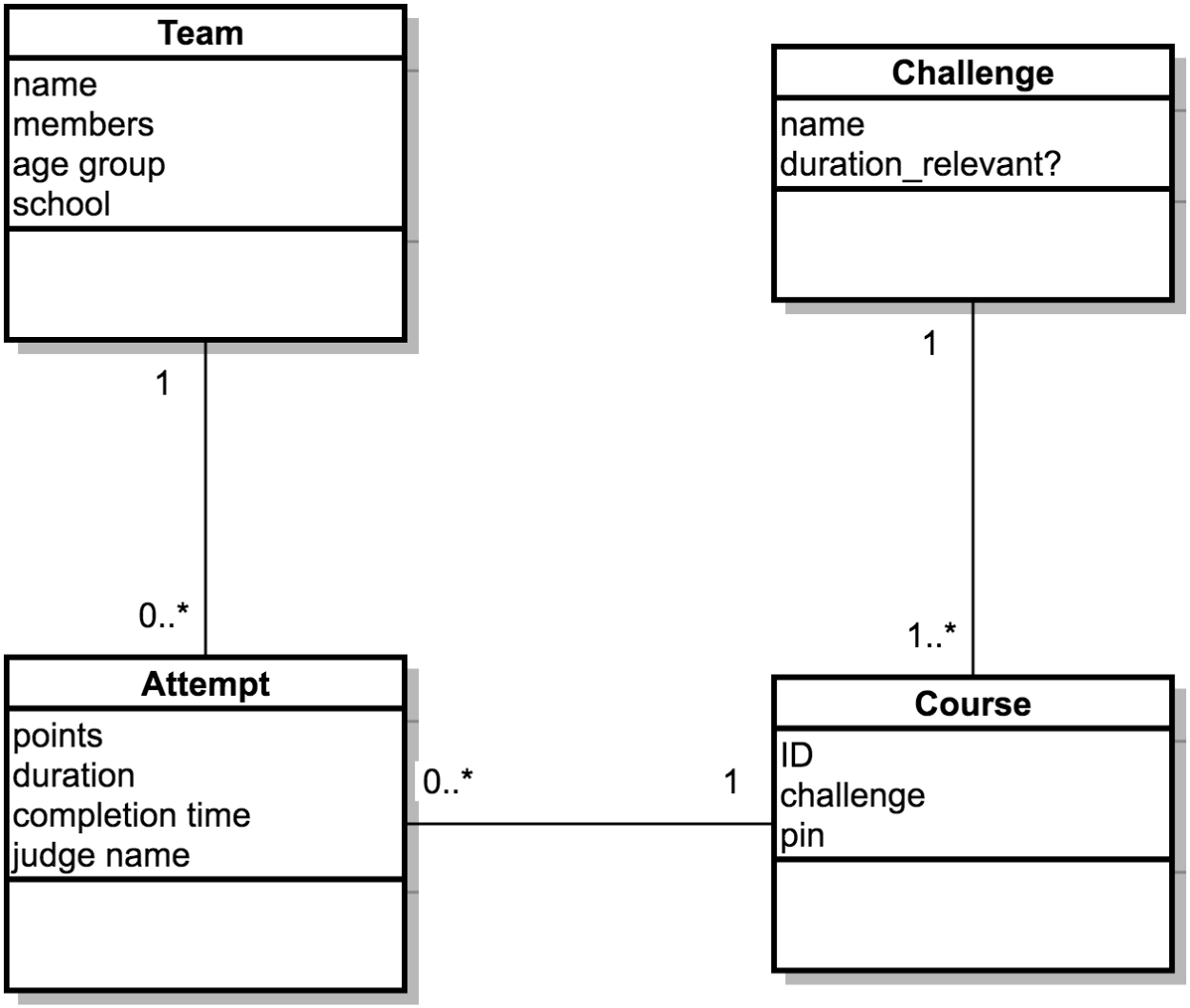


Figure 9: Class diagram for the relationships being modeled

3.3.1 Description
Figure 9 represents a class diagram of the core concept being modeled: that of teams taking attempts on courses for challenges. This model represents what will eventually be database tables in the final product.

3.4 Non-Functional Requirements

3.4.1 Performance

The application should respond quickly enough to avoid time-outs on default-configured web browsers. It is unacceptable for the application to time out while processing a request.

3.4.2 Reliability

The application should not lose data under any circumstances. The user should be given plenty of notice that data has or has not been recorded in the database.

3.4.3 Availability

The application should be fully available and responsive for the duration of the competition, which lasts 1 day every year. Availability during the competition should remain at 99.9%. 98% availability is acceptable for the rest of the year.

3.4.4 Security

The application should be resistant to casual attack by unskilled attackers. It should resist brute forcing password and similar attack. It does not have to be hardened against attack from corporate or state-level adversaries.

3.4.5 Maintainability

The application will be designed with the assumption that no maintenance will be available after delivery. The application will be well-written and

3.4.6 Portability

Application should be equally as usable on desktop and laptop computers as on mobile devices.

3.5 Design Constraints

The application must be usable on client-provided Android tablets.

4. Detailed Design

4.1 Architecture

The robotics scoring application will be built in Ruby on Rails using the Model-View-Controller Architecture. In this pattern, domain models are Models and all inherit from Rail's Object Relational Mapper, ActiveRecord. ActiveRecord models are backed by database tables; however, the database is not present in the diagram as all interactions with the database are done through ActiveRecord and no direct interface with the database from application code is necessary.

The web-facing interface is designed through objects called Controllers, which inherit from Ruby on Rails' ActionController class. Accessing different URLs calls different methods on the Controllers objects, also know as Controller actions. There is a programmer-defined “routing table” which maps URLs to controller actions.

Controllers return Views, which usually are procedurally generated HTML documents. Controllers can, with little configuration, be modified to change response type based on HTTP request type (a JSON request could return the JSON response of the model). Views are omitted from this document, being largely a graphical design problem.

In this document, methods from superclasses provided by Ruby on Rails' libraries are not present in the diagrams. Documentation for Rails superclasses can be found through the official Ruby on Rails documentation at api.rubyonrails.org

4.2 Detailed Class Diagrams

4.2.1 Models

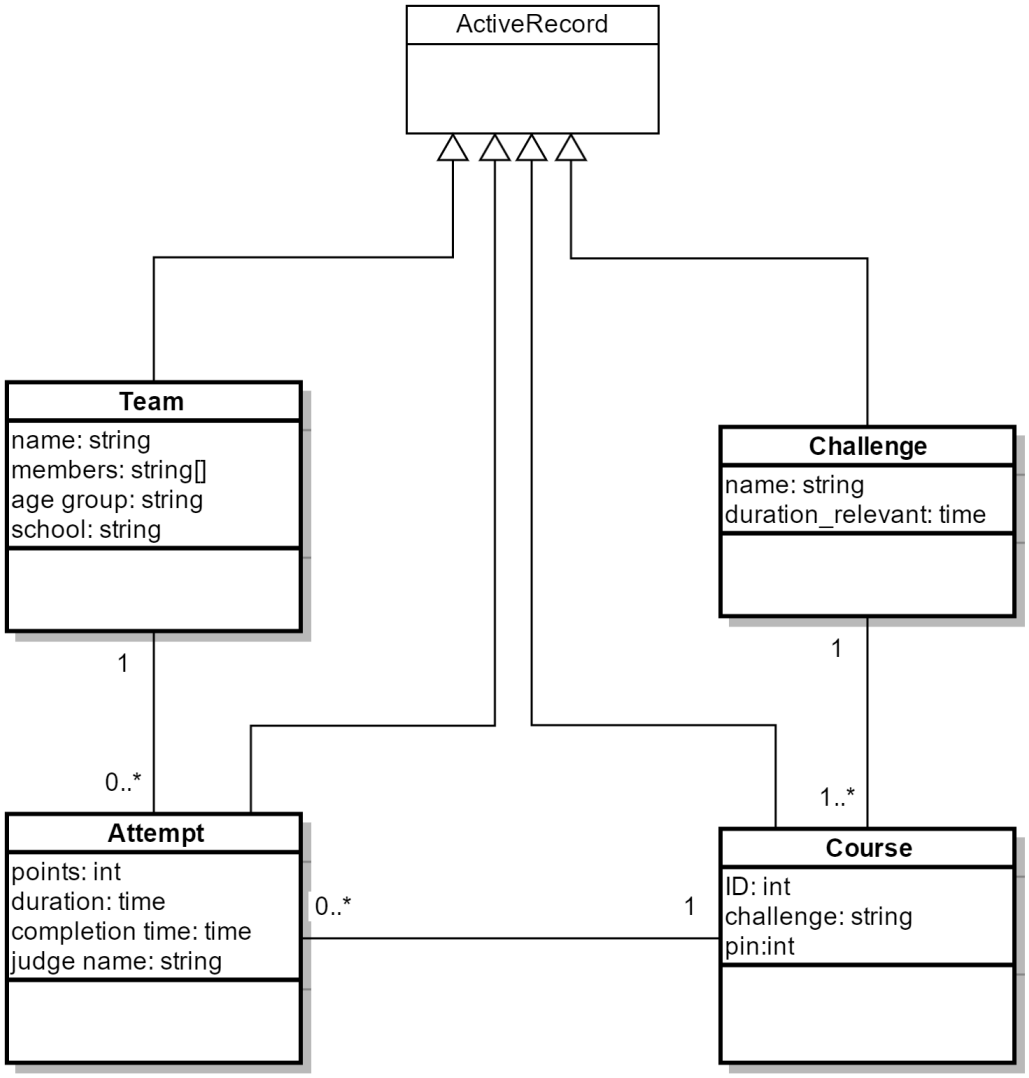


Figure 10: A detailed class diagram of the models.

4.2.1.1 Description

Models are Ruby classes responsible for communicating with the database, storing and validating data, and otherwise responsible for crunching numbers. The diagram has classes that define the scope of the problem: teams taking attempts on courses for challenges. This model diagram also represents what will be database tables. Note that there are no methods on the individual models; instead, all methods necessary are inherited from Ruby on Rail's ActiveRecord superclass, and no programmer-defined methods are required in this iteration of the project.

4.2.2 Controllers

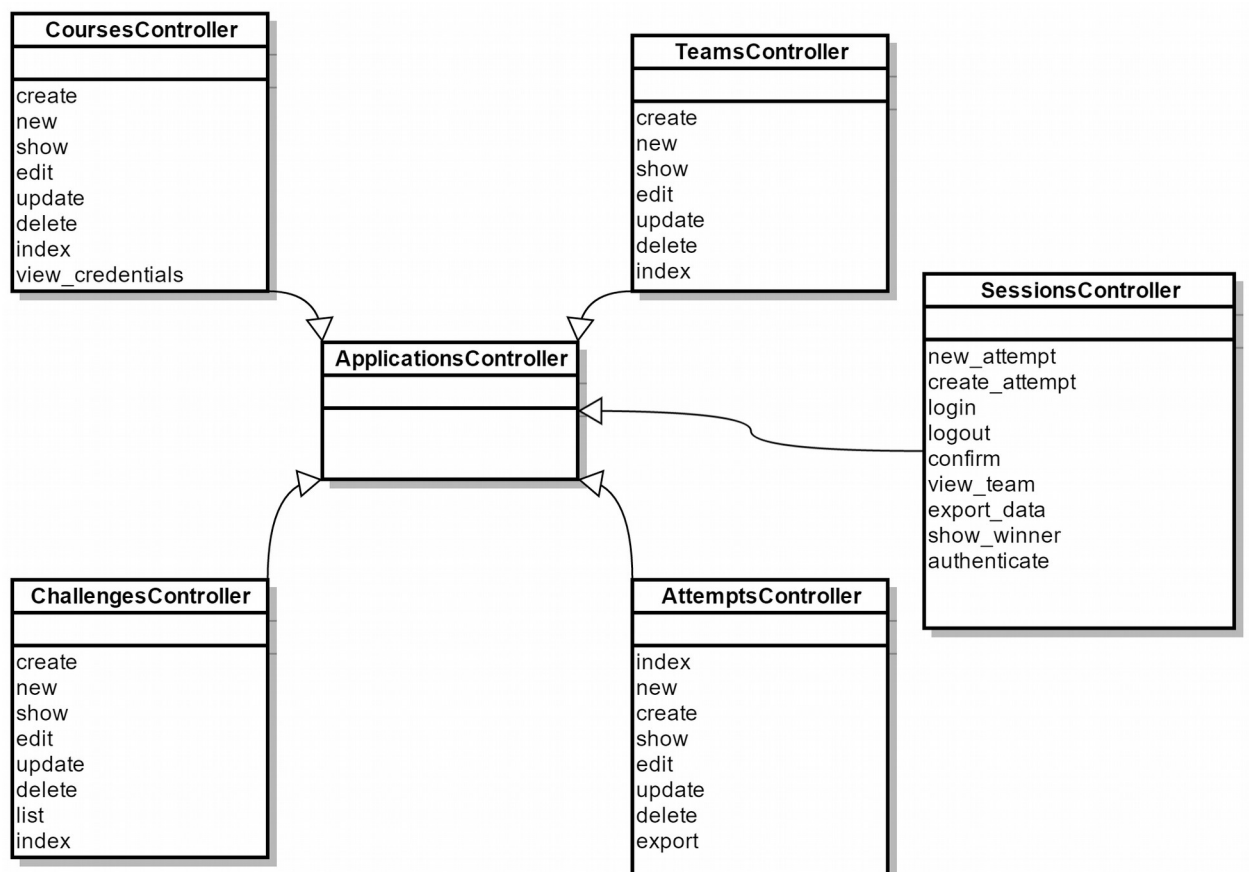


Figure 11: A detailed diagram of the controllers.

4.2.2.1 Description

Controllers are responsible for parsing user requests, data submissions, submissions, and the typical browser activity ultimately coordinating between the user, views and model. All of the controllers contain their respective methods and inherit from ApplicationController. ApplicationController inherits from ActionController, which is not diagramed and which provides the interface to Ruby on Rails.

4.3 Sequence Diagrams

4.3.1 Input Score

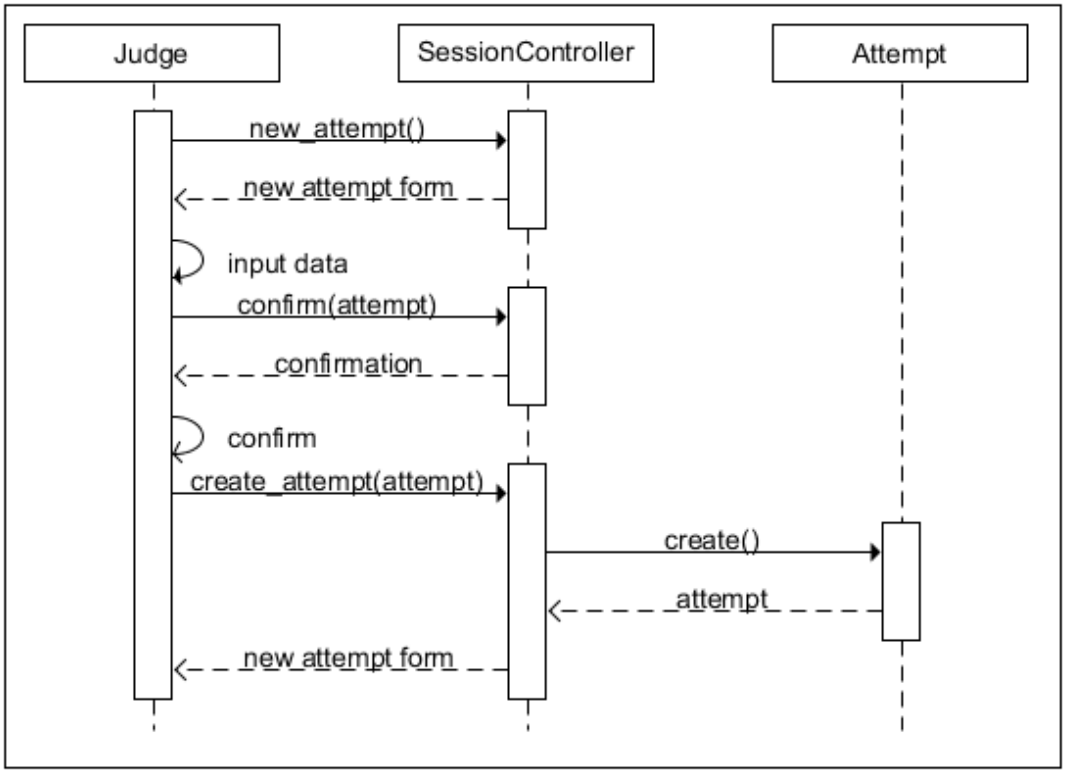


Figure 12: A sequence diagram for the input score use case.

4.3.1.1 Description
Figure 12 is a sequence diagram for the Use Case 3.2.1. First, a call to the SessionController#new_attempt is made, rendering the New Attempt form. The user inputs the data into this form and calls SessionController#confirm with the attempt data as parameters. This renders a confirmation page. The attempt score is confirmed and a call to SessionController#create_attempt with the attempt data as parameters creates a new Attempt object in the database. The object is returned and discarded, and the user is redirected back to the new attempt form.

4.3.2 Look Up Team Scores

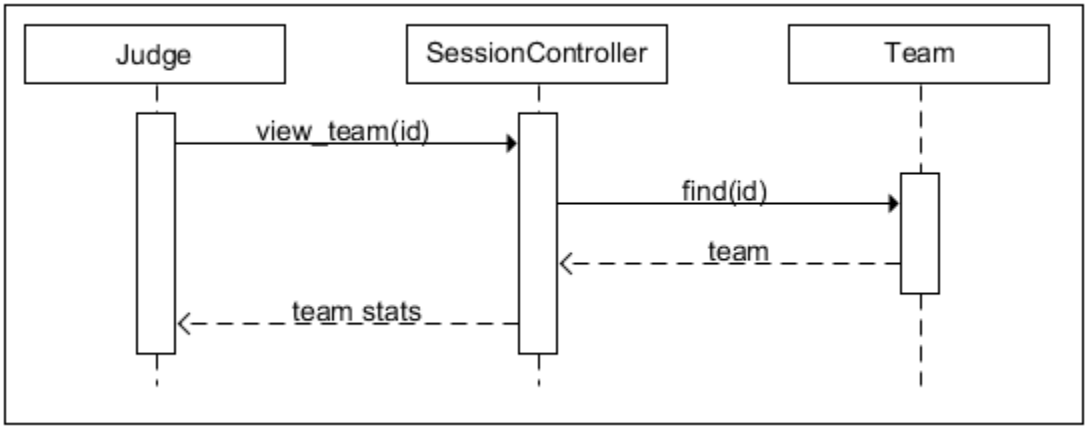


Figure 13: Sequence diagram for looking up team scores.

4.3.2.1 Description
Figure 13 is a sequence diagram for Use Case 3.2.2. First, the judge calls SessionController#view_team with the team ID as a parameter. The session controller calls Team.find with the team ID as the parameter, and the Team model returns the team with that ID. The team's stats are then returned to the judge.

4.3.3 Login

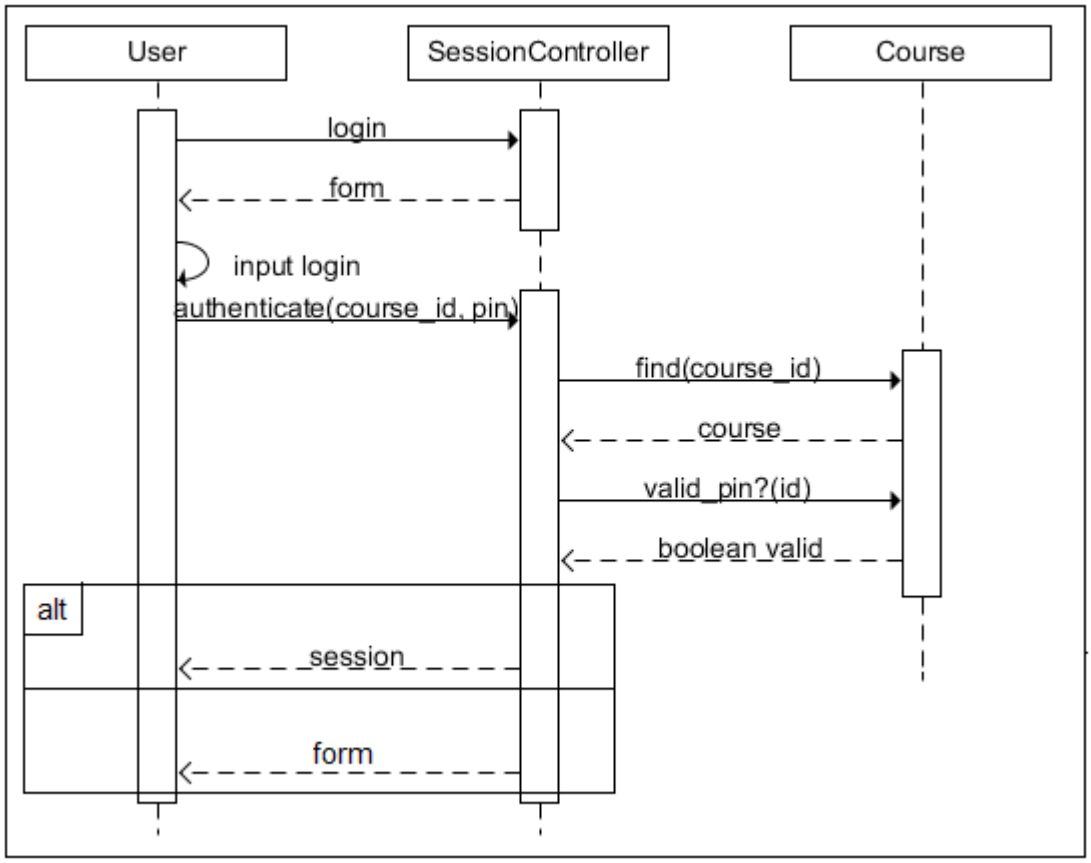
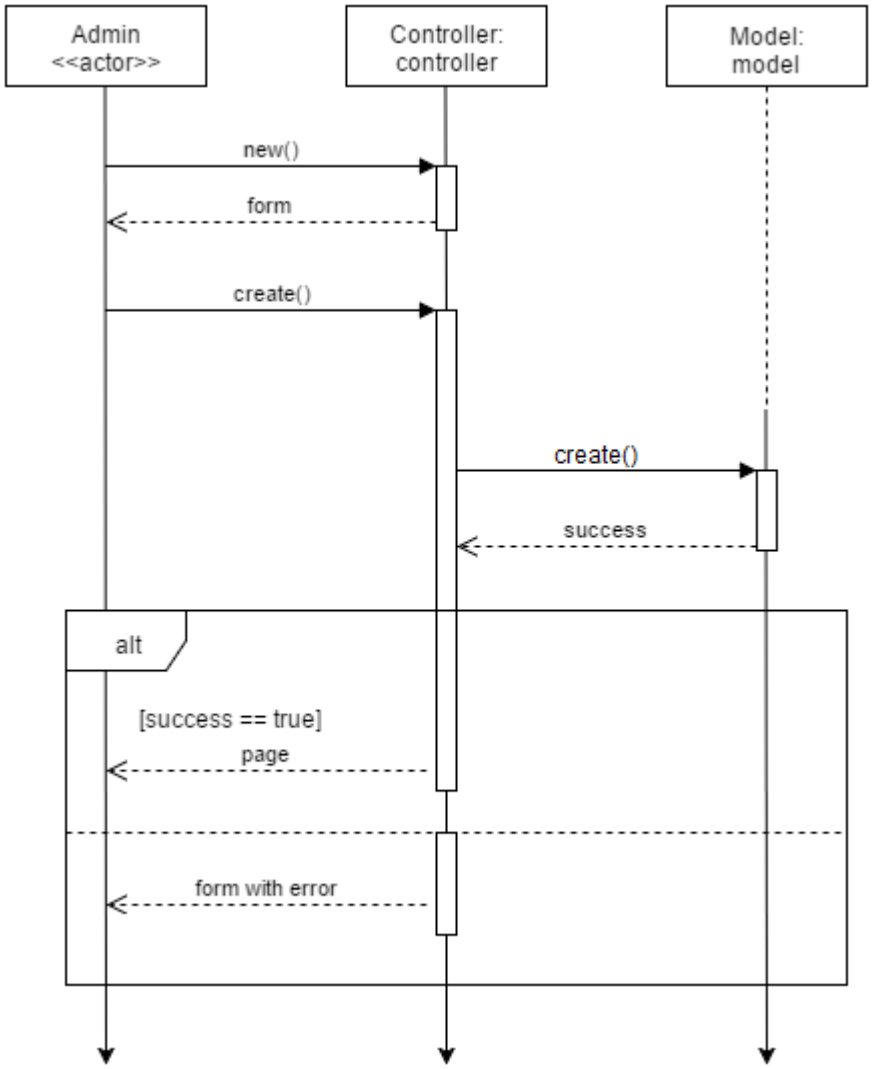


Figure 14: Sequence diagram for logging in.

4.3.3.1 Description

Figure 14 is a sequence diagram for Use Case 3.2.3. First, the user calls SessionController#login to get the login form. Then, the user inputs the login and calls SessionController#authenticate with the course ID and the login pin as parameters. The SessionController calls Course.find with the course ID and the Course model returns that course. The SessionController compares the course pin provided by the user to the one on the model. If pin is valid, the SessionController grants the user a session; if not, the user is kicked back to the login form.

4.3.4 Add Data

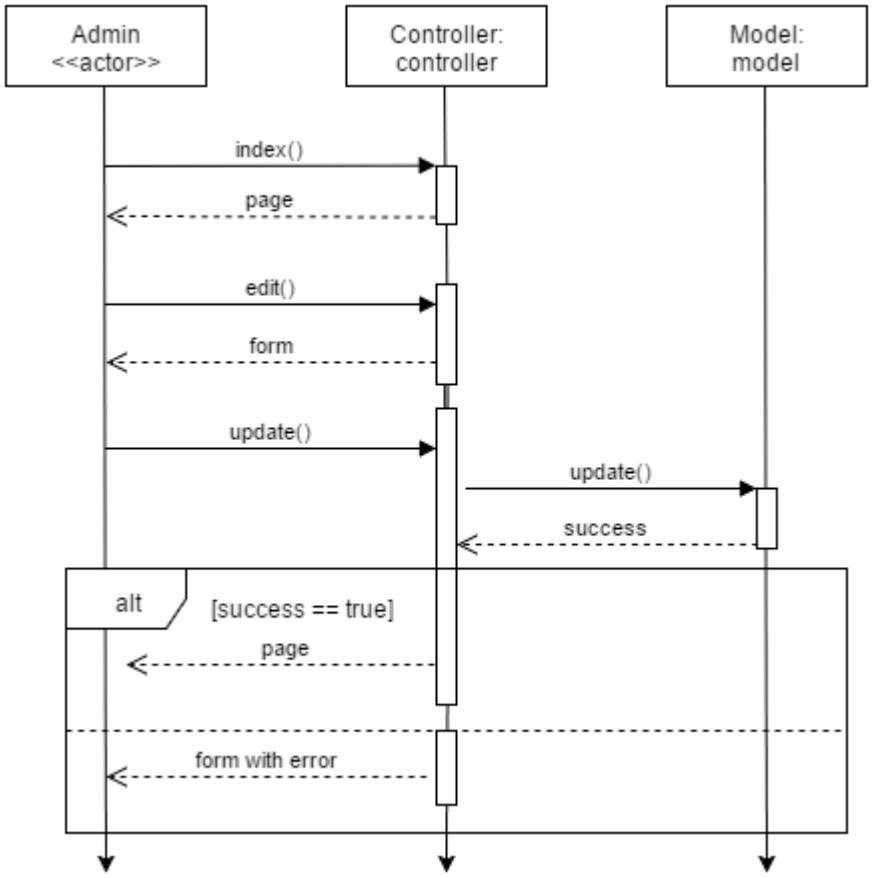


4.3.4.1 Description

This sequence diagram describes how the administrator interacts with the system in order to add data to the model. This includes Team, Attempt, Challenge, and Course data modified through the TeamsController, AttemptsController, ChallengesController, and CoursesController respectively.

When the administrator selects add from the view/edit data page for a particular Model, new() is called on the Controller. This returns a form with spaces for each piece of data the Model needs to represent. These correspond precisely to the Model class diagram and the public data members of the Model classes. The administrator fills out the form and submits it. This calls create() on the controller, which creates a new record. The record is saved to the model using create(), and a success boolean is returned. If the addition succeeded, true is returned, and the updated page is returned to the administrator. If the addition failed, false is returned, and a form with the error displayed is returned to the administrator.

4.3.5 Edit Data

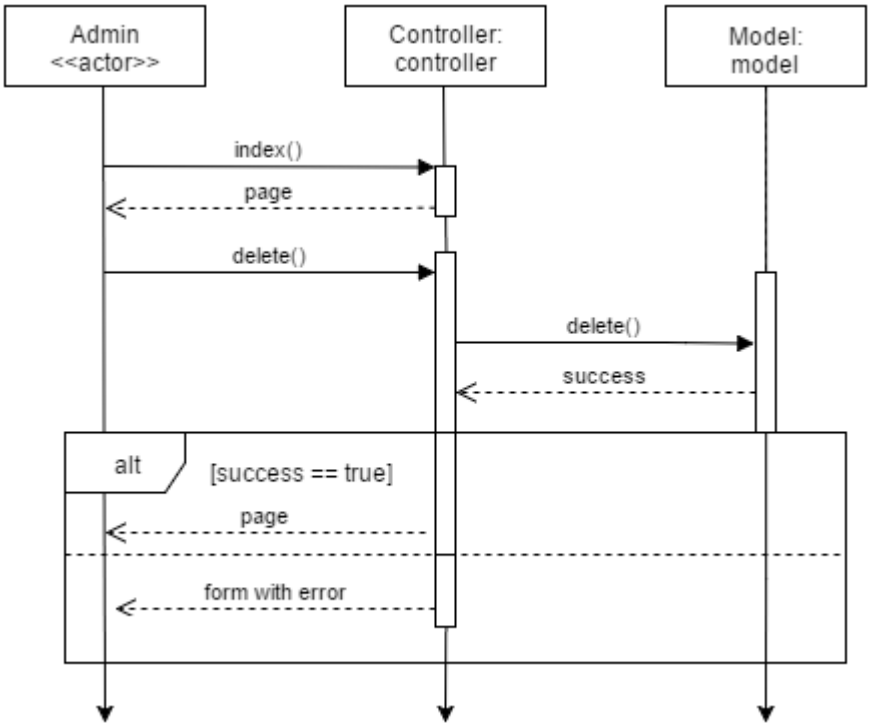


4.3.5.1 Description

This sequence diagram describes how the administrator interacts with the system in order to edit data in the model. This includes Team, Attempt, Challenge, and Course data modified through the TeamsController, AttemptsController, ChallengesController, and CoursesController respectively.

On loading the page, index() is called on the Controller. This returns a page containing all the data formatted into rows in a table. The administrator selects a row to edit. This calls edit() on the Controller, which returns a form with that record's data that can be edited. When the administrator has changed the data and submitted the form, update() is called on teh Controller. This calls update() on the Model and commits the changes. If the edit succeeded, true is returned, and the updated page is returned to the administrator. If the edit did not succeed, a form with an error message is returned to the administrator.

4.3.6 Delete Data

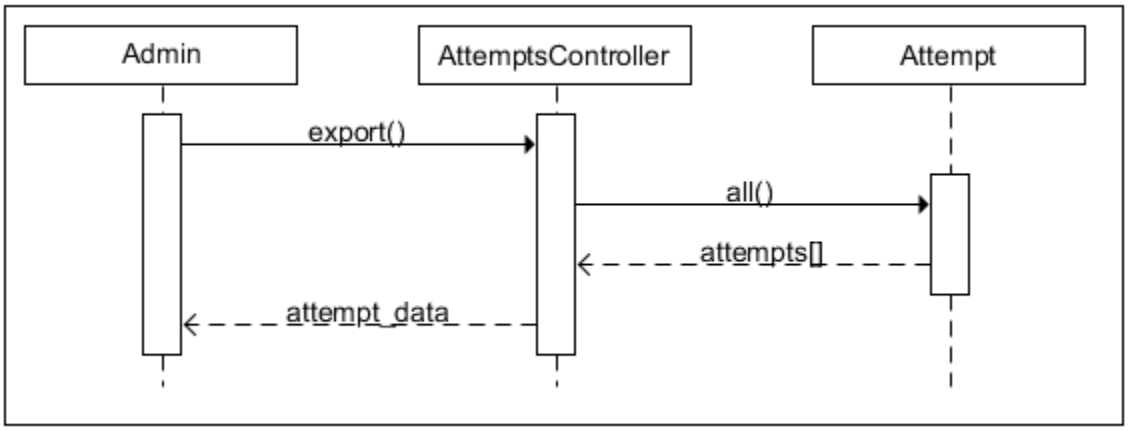


4.3.6.1 Description

This sequence diagram describes how the administrator interacts with the system in order to delete data from the model. This includes Team, Attempt, Challenge, and Course data modified through the TeamsController, AttemptsController, ChallengesController, and CoursesController respectively.

On loading the page, index() is called on the Controller. This returns a page containing all the data formatted into rows in a table. The administrator selects a row to delete and presses delete. This calls delete() on the Controller, which call delete() on the Model. The Model returns a boolean success value. If the deletion succeeded, then true is returned, and the updated page is returned to the administrator. If the deletion did not succeed, a form with an error message is returned to the administrator.

4.3.7 Export Data



4.3.7.1 Description
This sequence diagram shows how the administration can export all attempt data. The administrator calls AttemptsController#export, which calls Attempt.all, which returns a lits of all attempts, which is then turned by AttemptsController into a CSV, which is then returned to the user.

5. Change Management Process

If this document requires changes, changes are proposed to the team. If accepted, the changes are made and a short summary of the changes made are recorded in the log. The version after the change is applied is recorded in the change log. The new document is saved in a file named with the version number appended, as both a ODT file and PDF file. No old versions are to be deleted.

The version is adjusted according to the following system:

1. Any changes that substantially alter existing content in the document shall increment the major version, the first number in the three dot-separated components of the version
2. Any changes adding new content but not substantially altering the existing content of the document shall increment the minor version, the second component of the version number.
3. Any changes altering minor details in wording, spelling, grammar, etc., that do not alter the semantics of the document shall increment the patch version, the third component of the version number.

Appendix A: Routing Table

HTTP Verb	Route	Controller Action	Used For
Session Controller			
GET	/session	session#login	Display the login form for a session, or drop the user to the attempt input page if already logged in
POST	/session	session#authenticate	Check that the user's provided password for a course matches the database
DELETE	/session	session#logout	End a judging session
GET	/session/attempt	session#new_attempt	Display the form for judges to input new attempt data
GET	/session/attempt/confirm	session#confirm	Display a page with the attempt data and provide a confirmation screen for the team captain
POST	/session/attempt	session#create_attempt	Create a new confirmed attempt
GET	/session/team/:id	session#view_team	Show data for a team
GET	/session/winners	session#winners	Show the winners in each age group
Challenges Controller			
GET	/challenges	challenges#index	Display all challenges
GET	/challenges/new	challenges#new	HTML form to create a new challenge
POST	/challenges	challenges#create	Create new challenge
GET	/challenges/:id	challenges#show	Display a specific challenge
GET	/challenges/:id/edit	challenges#edit	HTML form for editing a challenge
PATCH/PUT	/challenges/:id	challenges#update	Update a challenge
DELETE	/challenges/:id	challenges#delete	Delete a specific challenge

HTTP Verb	Route	Controller Action	Used For
Attempt Controller			
GET	/attempts	attempts#index	Display all attempts
GET	/attempts/new	attempts#new	HTML form to create a new attempt
POST	/attempts	attempts#create	Create new attempt
GET	/attempts/:id	attempts#show	Display a specific attempt
GET	/attempts/:id/edit	attempts#edit	HTML form for editing an attempt
PATCH/PUT	/attempts/:id	attempts#update	Update an attempt
DELETE	/attempts/:id	attempts#delete	Delete a specific attempt
GET	/attempts/export	attempts#export	Get all raw attempt data
Team Controller			
GET	/teams	teams#index	Display all teams
GET	/teams/new	teams#new	HTML form to create a new team
POST	/teams	teams#create	Create new team
GET	/teams/:id	teams#show	Display a specific team
GET	/teams/:id/edit	teams#edit	HTML form for editing a team
PATCH/PUT	/teams/:id	teams#update	Update a team
DELETE	/teams/:id	teams#delete	Delete a specific team
Course Controller			
GET	/courses	courses#index	Display all courses
GET	/courses/new	courses#new	HTML form to create a new course
POST	/courses	courses#create	Create new course
GET	/courses/:id	courses#show	Display a specific course
GET	/courses/:id/edit	courses#edit	HTML form for editing a course
PATCH/PUT	/courses/:id	courses#update	Update a course
DELETE	/courses/:id	courses#delete	Delete a specific course