Software Requirements

Playmaker App

Version 1.23

Austin Beattie
Brock Matter
Daniel Redmond

Revision History

Date	Description	Author	Version
1-23-2016	Created initial SRS document	Austin Beattie	1.0
1-29-2016	Updated project description and initial features	Austin Beattie, Brock Matter, Daniel Redmond	1.1
2-12-2016	Added functional requirements	Daniel Redmond	1.2
2-12-2016	Added more functional requirements	Brock Matter	1.3
2-13-2016	Edited functional requirements, added non-functional	Austin Beattie	1.4

	requirements		
2-13-2016	Added initial activity diagrams and Use Case Diagram	Daniel Redmond	1.5
2-15-2016	Added more activity diagrams	Brock Matter	1.6
2-15-2016	Updated Functional Requirements	Austin Beattie	1.7
2-16-2016	Added final Use Case Diagram, more activity diagrams	Austin Beattie	1.8
2-16-2016	Added Class Diagram	Daniel Redmond	1.9
2-16-2016	Final revisions to SRS	Austin Beattie, Daniel Redmond, Brock Matter	1.10
3/2/16	Updated Class Diagram	Daniel Redmond	1.11
3/2/16	Updated FRs	Brock Matter	1.12
3/3/16	Updated FRs, NFRs, Intro	Austin Beattie, Brock Matter	1.13
3/8/16	Updated Class Diagram	Austin Beattie	1.14
3/30/16	Added Sequence Diagram Section	Daniel Redmond	1.15
3/30/16	Renumbered sections and diagrams. Updated TOC	Daniel Redmond	1.16
3/30/16	Added Seq. diagrams	Austin Beattie, Daniel Redmond, Brock Matter	1.17
3/30/16	Added detailed class diagram	Austin Beattie	1.18
4/5/2016	Added revised sequence diagrams	Austin Beattie, Daniel Redmond, Brock Matter	1.19
4/5/2016	Updated descriptions for class diagrams, sequence diagrams	Daniel Redmond	1.20
4/5/2016	Updated TOC	Austin Beattie	1.20
4/5/2016	Updated select sequence diagrams and descriptions	Brock Matter	1.21
4/5/2016	Added revised detailed class diagram	Austin Beattie	1.22
4/5/2016	Final editions.	Daniel Redmond	1.23

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements for the Playmaker Application. Within this document are a general description of the application, an overview of the application's functions, all functional and nonfunctional requirements, and diagrams representing primary use cases and functionality of the application.

1.2 Scope

The Playmaker App will provide an intuitive and modern platform for football teams to create, manage, and share football playbooks. Users will have simple interfaces for play creation and editing, playbook management, and team management.

1.3 Key Terms

- ***App*** - football playmaker application
- ***Block*** - an attempt by a football player to block another player from proceeding in a certain direction
- ***Coach*** - a coach or leader of a football team, one of the target users of the app
- ***Coach profile*** - in-app profile representing a coach
- ***Editor*** - the tool within the app for creating and modifying football plays
- ***Football*** - American football
- ***Play animation*** - working term for the animation of a play diagram (i.e. players represented in the diagram move according to their routes/runs/blocks/coverage schemes, roughly simulating the timing of the play in real life)
- ***Play*** - a single football play (i.e. in American football, a plan or strategy for the moving of the football downfield, or for defending against such a plan); also used to mean *play diagram*
- ***Playbook*** - a player's or coach's personalized set of plays stored within the app; also a physical football playbook
- ***Play diagram*** - the representation of a football play within the app
- ***Play editor*** - see editor
- ***Player*** - either a real football player or the representation thereof within the app
- ***Player profile*** - in-app profile representing a player
- ***Playmaker [app]*** - the current working title for the app
- ***Project*** - the design project for which the app is being developed
- ***Route*** - a path followed by a football player of a certain position in a given play
- ***Run*** - a path followed by a football player of a certain position who is carrying the ball in a given play
- ***Tag*** - a short phrase describing an element of a play, used for searching through plays

- ***Team*** - a set of app users with a shared playbook (usually represents a football team in real life); also a real life football team
- ***User*** - target user of the app, usually either a football player or coach
- ***User Profile*** - see user account
- ***User Account*** - an account created by a user containing all the user's information.
- ***View*** - generally, 'view' is used in this document to describe an interface within the app that has a particular function (e.g. the 'play view' is the interface which displays a single play and gives the user options for that play, while the 'editor view' is the interface for play editing and creation)

2. Project Description

2.1 Overview and Target Users

The intent of this project is to create a smart phone application (app) which can be used as a complete football playbook and learning tool by football players and coaches of all ages. The app will allow for quick and simple creation and sharing of football plays, as well as basic tools for creating and managing teams of users with shared playbooks.

Currently, football coaches and players at all levels use paper playbooks which must be consistently updated and maintained via printing and reprinting of physical pages, usually contained in binders or folders. In order for players to study plays, they must carry these binders with them wherever they are. Updating plays or adding new plays requires printing them out and distributing them to all players physically (or having each player print them individually). For many youth football teams this process can be inconvenient, and for many casual players of intramural, flag, or neighborhood football it is impractical. For both, it can be difficult to capture the dynamic nature of football plays in small, static pictures.

Thus, the target users for the app fall into two categories: casual football players who do not care to print real playbooks, and official football teams who wish to streamline the whole process of creating, maintaining, and learning playbooks.

2.2 Application Features

2.2.1 Play Creation and Editing

The application will allow users to graphically create plays, dragging and dropping players onto a play diagram then drawing their routes, blocks, runs, or coverage zones. Once a play is created, it can be categorized, tagged, saved to one or more playbooks, and shared with team members.

2.2.2 Playbook

The application will allow users to create and manage custom collections of plays called playbooks. The playbook interface will provide a graphic list of play previews (miniature versions of each play's diagram) and a menu to display or search for plays by tag. The playbook interface will also allow users to select one or more plays and perform actions on them (e.g. add tags to plays, delete plays, edit a play, share plays, copy plays to other playbooks, create a quiz for plays, or change play categories). While in the playbook, a user can select any play for viewing in the play viewer interface.

2.2.3 Play viewer

The application will provide an interface to view individual plays for study. The play viewer displays a single play diagram and information about the play. The user will have access to several actions in the play view: open the current play in the editor, share the current play, create a quiz based on the current play, or display similar plays in the playbook. The user will also be able to view play animations within the play viewer.

2.2.4 Team manager

The application will provide an interface for viewing managing all teams of which a user is a player and coach. For teams the user coaches, the user can add, remove, and edit other player and coach profiles. For teams the user does not coach, the user can view profiles of other players and coaches on the team. The interface will allow users to create new teams and access playbooks for each team or player.

2.2.5 User profile creation/management

Each user must create a profile in order to create a team or be added to a team. The application will provide an interface for creating and modifying a user's profile. Profiles include a username, password, the user's real name, the user's email address, a list of teams for which the user is a coach, and a list of teams for which the user is a player. For each team on which the user is a player, the user can select a position and number, if desired.

2.2.6 User login

In order to share plays or be part of a team, a user must log in to a user profile. The application will provide a login interface which prompts a user for a username or email address and a password.

3. Requirements

3.1 Requirement Specifications

Each requirement defined in this section contains a short description of the requirement, and descriptions of the input, output, processing, and error handling for the requirement, if they are applicable.

3.2 Functional Requirements

3.2.1 Login

- *Description*
  The user can log in to the system.

- *Input*
  The username and password.

- *Processing*
  The username and password are checked against the database to determine if login credentials are valid.

- *Output*
  The user is logged into the system.

- *Error Handling*
  If the username or password is incorrect, the user is notified and asked to re-enter login credentials.

3.2.2 Create Account

- *Description*
  A new user can create an account.

- *Input*
  Name, username, and password are required to create an account. Username and password must be at least 6 characters.

- *Processing*
  A new account is created in the database with the given name, username, and password.

- *Output*
  New user account.

- *Error Handling*
  If the username already exists, no account will be created and the user will be notified. If username or password is invalid (not 6 or more characters), no account will be created and the user will be notified.

3.2.3 View Playbook

- *Description*
  A user must be able to view all plays in a playbook

- *Input*
  User selects playbook to view.

- *Processing*
  The selected playbook is loaded from the database.

- *Output*
  The playbook is displayed on the screen.

- *Error Handling*
  If the playbook has no plays, then the user will be notified.

3.2.4 Search Playbook

- *Description*
  A user can search a playbook for a play by name or tags.

- *Input*
  The user types a name or tag in the search bar

- *Processing*
  The database is searched for any plays matching the search query.

- *Output*
  Matching plays are displayed in the playbook view. If no plays are found, a message is displayed which says “No Plays Found”

- *Error Handling*
  None

3.2.5 Display Plays by Tag

- *Description*
  A user can view plays with a given tag or tags

- *Input*
  User selects tag(s) from list of existing tags in playbook view

- *Processing*
  Database is queried for all plays with selected tag

- *Output*
  Plays with selected tag are displayed on screen

- *Error Handling*
  If no plays are associated with a tag the database will not return any plays upon query.

3.2.6 Create Team

- *Description*
  A user can create a team with a unique team name.

- *Input*
  Unique team name.

- *Processing*
  A new team is created in the database.

- *Output*
  New team.

- *Error Handling*
  If the team name is already taken, no new team will be created and the user will be notified.

3.2.7 Add Team Member

- *Description*
  A coach can add members to his team. The members of the team can view the team's playbook

- *Input*
  New team member username.

- *Processing*
  A request to join the team is sent to the user associated with the entered username.

- *Output*
  The user is notified if the request was successfully sent.

- *Error Handling*
  If the username is invalid, no request is sent and the user is notified.

3.2.8 Reply to Team Request

- *Description*
  A user gets a request to join a team and either joins or rejects the request.

- *Input*
  The user is presented with two options: join team or reject request.

- *Processing*
  If the user joins the team, then the user is added to the team database. If the user rejects the request, the request is deleted.

- *Output*
  If the user accepts the request, they are taken to the team page.

3.2.9 Leave Team

- *Description*
  A user can leave a team.

- *Input*
  The user selects the team they want to leave.

- *Processing*
  The user is removed from the team's database.

- *Output*
  Confirmation of leaving the team.

3.2.10 Delete Team

- *Description*
  A coach can delete a team.

- *Input*
  The coach selects the team to delete.

- *Processing*
  The team database is deleted.

- *Output*

Confirmation of deletion.

3.2.11 Remove Team Member

- *Description*
  A coach can remove a team member.

- *Input*
  Username of the team member to removed.

- *Processing*
  The user with the associated username is removed from the team database.

- *Output*
  Confirmation is given to the user.

- *Error Handling*
  If the username is incorrect, then the coach should be notified.

3.2.12 Create Play

- *Description*
  A user can create a play, if the user has coach permissions for current team.

- *Input*
  User inputs play name, selects whether the new play is offensive or defensive, then edits the play by dragging and dropping player icons onto the play diagram and drawing offensive or defensive actions for each player. While editing the play, the user can also select tags for the play, and add additional text information to the play.

- *Processing*
  The new play, if valid, is saved to the database and added to the playbook with tags

- *Output*
  Upon attempt to save, user is notified whether the play was valid, and if true that the play was saved.

- *Error Handling*

If play was not valid (ex. too many players on line of scrimmage), then the user is returned back to play creation interface.

3.2.13 Edit play

- *Description*
  The user can edit all elements of an existing play, if user has coach permissions.

- *Input*
  The user modifies the players, actions, info, or tags of the play in the play editor and chooses to save the play.

- *Processing*
  The play is updated in the database and playbook.

- *Output*
  The app returns the user to the play view for the updated play.

- *Error Handling*
  If play name is invalid, user is prompted for new name.

3.2.14 Play View

- *Description*
  The app must provide an interface to view a single play and access all options relating to the play.

- *Input*
  The user selects a play in the playbook, show as a scrolling view of minimized play diagrams with accompanying descriptions and list of tags.

- *Processing*
  The details of the play are loaded from the database.

- *Output*
  The play diagram and information are displayed in the play view.

3.2.15 Manage Playbook

- *Description*
  The user can create plays, delete (one or more) plays, open a single play for viewing, or open a play for editing from within the playbook view. A user with ‘player’ permissions can only open a play for viewing.

- *Input*
  The user selects a play or plays and an action to carry out on the play(s).

- *Processing*
  The system deletes plays or opens one play for editing or viewing.

- *Output*
  If a single play is opened for viewing or editing, the system displays the play view or editor.

3.2.16 Create Playbook for Team

- *Description*
  A user can create one or more playbooks for each team the user coaches.

- *Input*
  User selects a team and the option to add a playbook, and enters a name for the playbook.

- *Processing*
  The playbook is created in the system and added to the team’s list of playbooks.

- *Output*
  The new playbook is displayed.

- *Error Handling*
  If playbook name is already in use, the user will be prompted to enter a new name.

3.2.17 Delete Playbook

- *Description*
  A user must be able to delete playbooks associated with teams which the user coaches.

- *Input*

User selects a playbook and to delete.

- *Processing*
  The selected playbook is deleted from the database.

- *Output*
  The app displays the updated list of playbooks.

3.2.18 Rename Playbook

- *Description*
  A user must be able to rename a playbook.

- *Input*
  User selects playbook and enters a new name.

- *Processing*
  The playbook is updated in the system database.

- *Output*
  The updated playbook list is displayed.

- *Error Handling*
  If the playbook name is already in use for current team, the user must enter a new name.

3.2.19 Update User Account

- *Description*
  A user can update information in their user account profile (username, password, team permissions).

- *Input*
  User inputs new information for user account.

- *Processing*
  User account information is updated in system database.

- *Output*
  The updated user profile is displayed.

- *Error Handling*
  If the user enters invalid information, the changes to the profile will not be saved.

3.2.20 Delete User Account

- *Description*
  A user must be able to delete their own user account from the system.

- *Input*
  User selects “Delete User Account.”

- *Processing*
  The User Account is removed from the system database.

3.2.21 View User Profile

- *Description*
  A user must be able to view his user profile.

- *Input*
  The selects “View profile.”

- *Processing*
  The system loads user profile data.

- *Output*

The user profile is displayed.

- *Error Handling*
  If there is no internet connection to load the user profile, then the user will be notified to connect to the internet.

3.2.22 User Permissions

- *Description*
  A user must have either player or coach permissions for each team he is part of. A player has limited permissions in that they are not able to manage a team, manage a playbook, or edit plays as the standard permissions. A coach has full permissions for their account as well as the ability to grant play editing permission to a player.

3.2.23 Modify User Permissions

- *Description*
  A team coach can modify user permissions of players.

- *Input*
  Coach user sets permissions for team player.

- *Processing*
  User account information is updated in system database.

- *Output*
  Success message.

3.3 Non-functional Requirements

3.3.1 Security

- *Description*
  A user should not be able to access any plays or team information for teams the user is not part of, and should not be able to access any user account information without the proper username/email and password.

3.3.2 Performance

- *Description*
  The application should quickly respond to user input. Drawing plays should be responsive and fluid. Changes to the team playbook should be available to all teammates within 1 minute.

3.3.3 Reliability

- *Description*
  No data should be lost. If the app does not function correctly, users should be notified.

3.3.4 Offline Availability of Plays

- *Description*
  Plays should be maintained in a local database for access offline. New plays added by other team members or coaches will not be available until an internet connection exists.

3.3.5 Play Animations

- *Description*
  A user should be able to view animated versions of plays.

- *Input*
  User presses animate button while in play view.

- *Processing*

System runs animation for selected play.

- *Output*
  System displays animated version of play.

3.4 Use Case and Activity Diagrams

3.4.1 Use Case Diagram

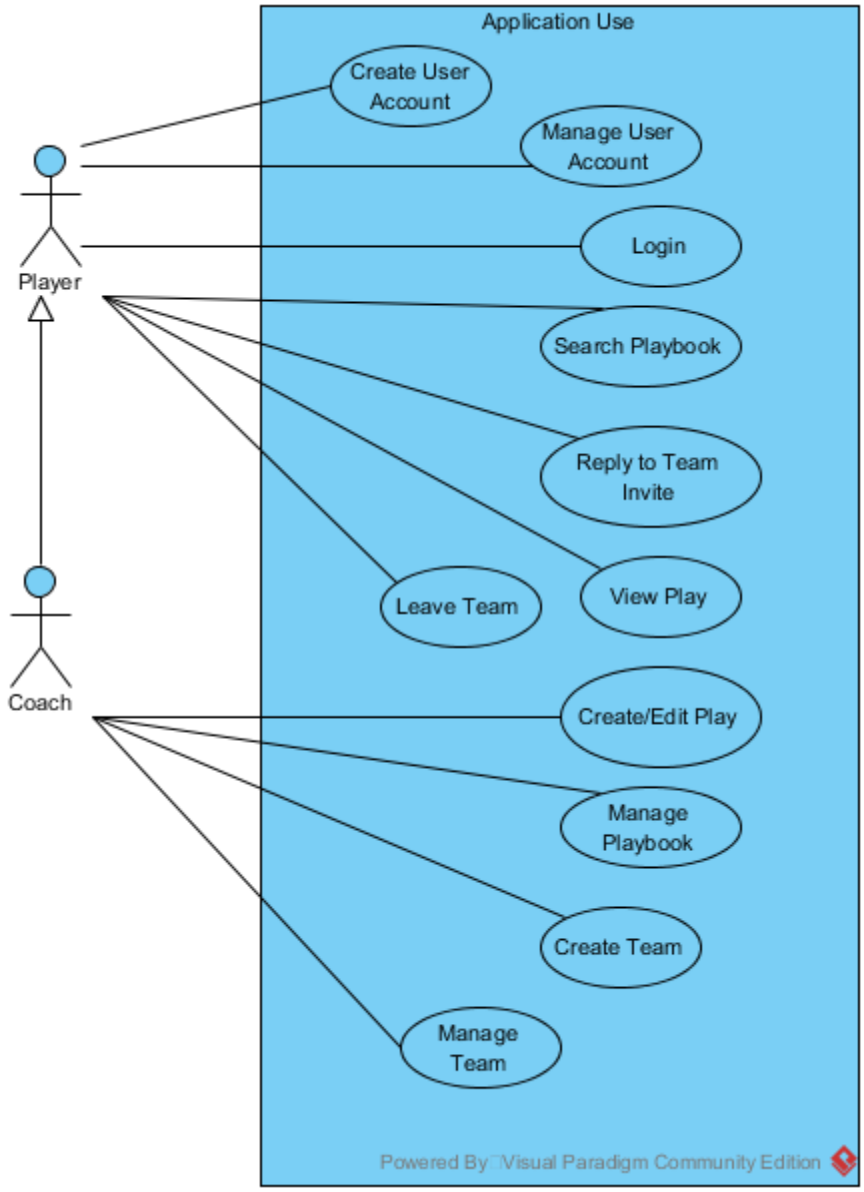

Figure 1: Use Case Diagram for Playmaker Application

Description

There are 12 use cases defined for the system, with two possible types of actors: player and coach. A coach can perform all actions a player can perform, with the addition of actions relating to team and playbook management. A given user can act only as a player for teams the user does not coach, but can act as a coach for teams the user

creates and/or coaches. The specific use cases for both players and coaches are outlined in the remainder of this section.

3.4.2 Create Team

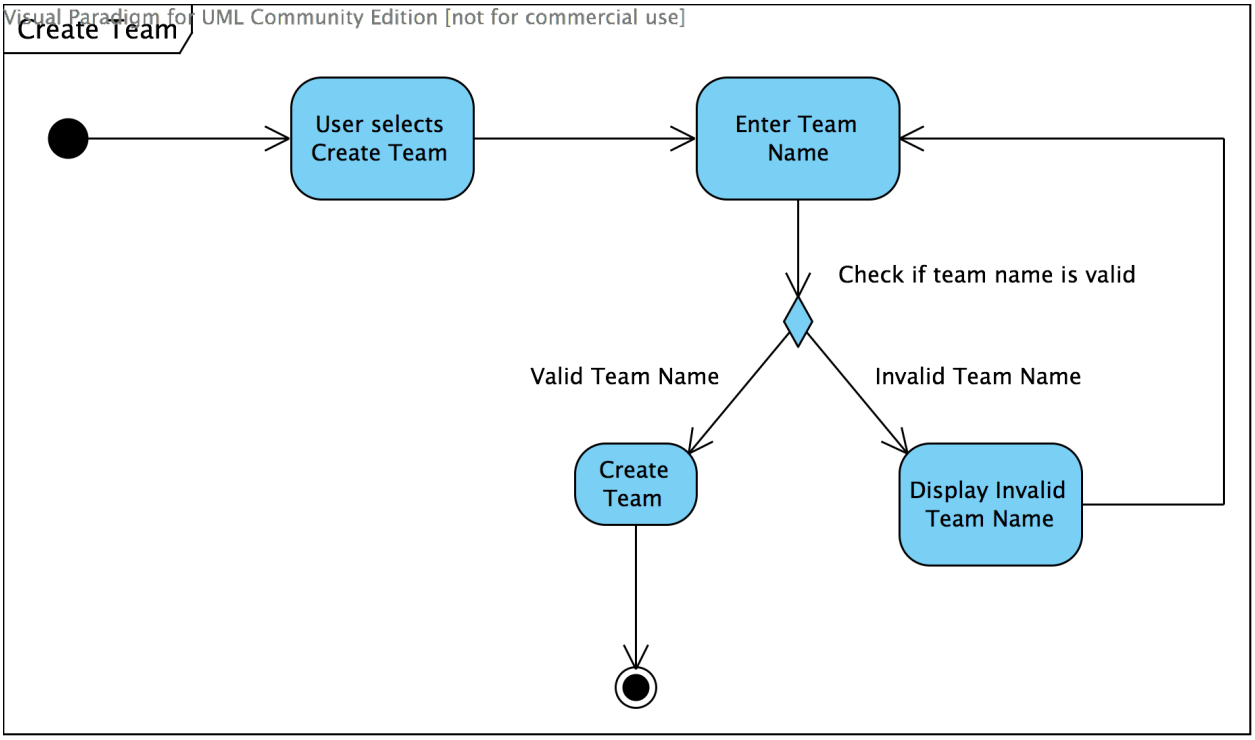

Figure 2: Activity diagram for "Create Team" use case

- *FR 3.2.6 Create Team*
- *Description*
  Figure 2 shows how the user will create a new team. The user will enter a team name that must be unique. If the team name is not already in the system and consists of valid characters, then the team is created in the system. If the team name is invalid, then the user is notified and asked to use a different name.

3.4.3 Manage Team

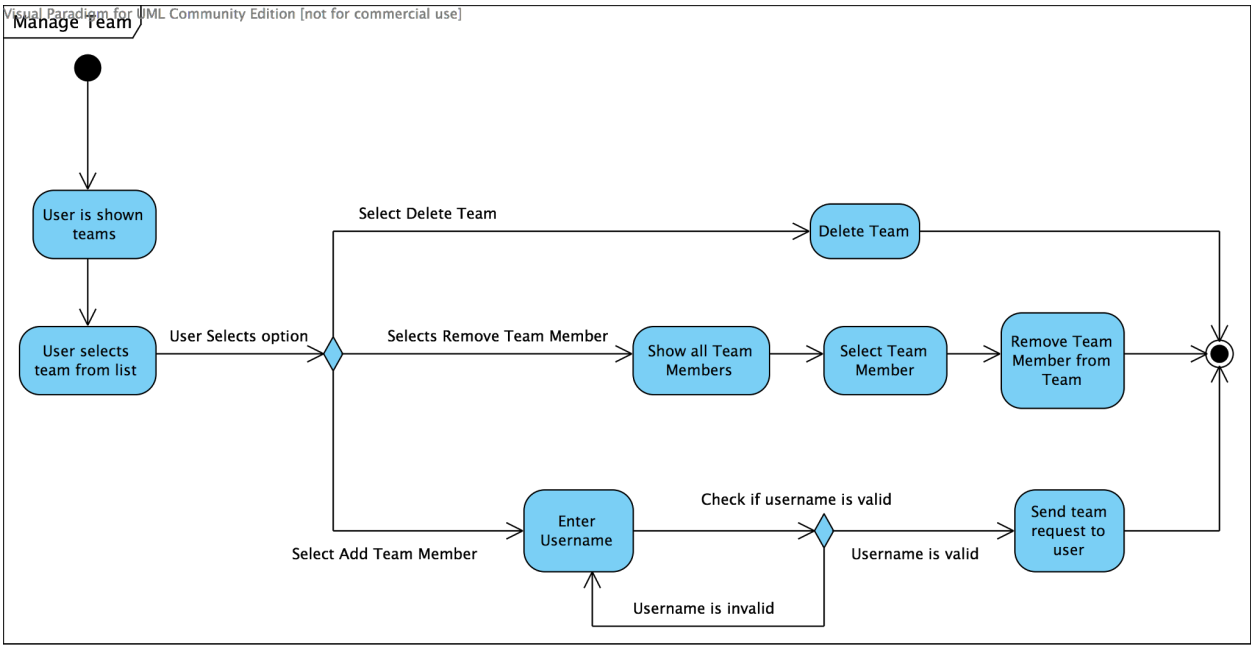

Figure 3: Activity Diagram for "Manage Team"

- *FR 3.2.7 Add Team Member, 3.2.10 Delete Team, 3.2.11 Remove Team Member, FR 3.2.22 User Permissions, FR 3.2.23 Update User Permissions*
- *Description*

  Figure 3 shows how users can manage their teams. First, users are shown all of their teams. The user then selects the team they want to edit. Then the user is given options to add team member, remove team member, or delete the team. If the user selects add team member, they enter a username and the system sends a request to join the team to that user. If the user selects remove team member, then all the team members are shown and the user selects the one to delete. If the user selects delete team, then the system deletes the team.

3.4.4 Reply to Team Request

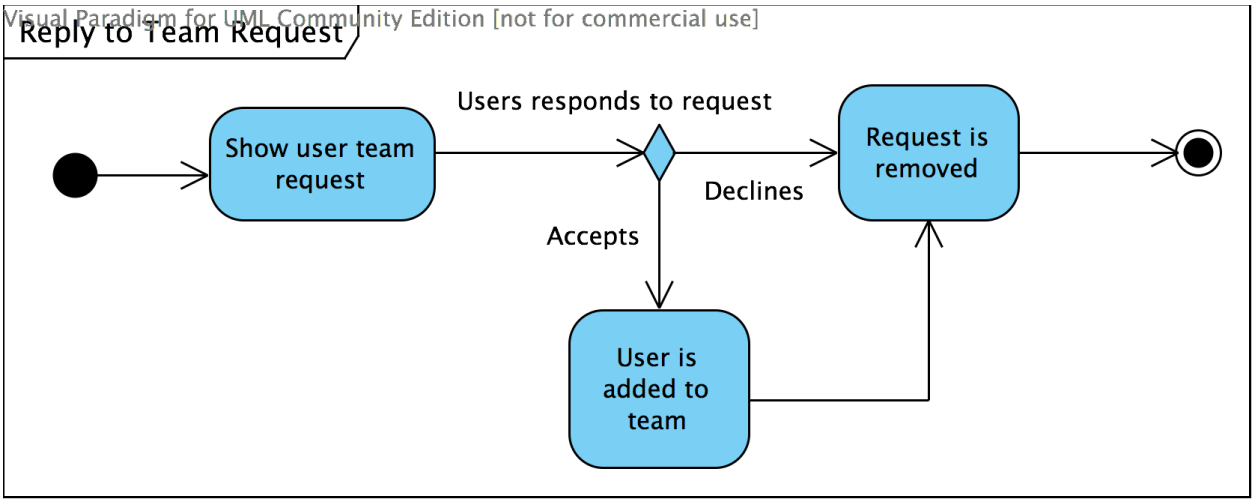

Figure 4: Activity Diagram for "Reply to Team Request" Use Case

- *FR 3.2.8 Reply to team Request*
- *Description*
  Figure 4 shows how the user can reply to a team request. The user receives an invite to join a team. Then the user can either accept the invite, which adds them to the team, or declines the invitation, which removes the request.

3.4.5 Leave Team

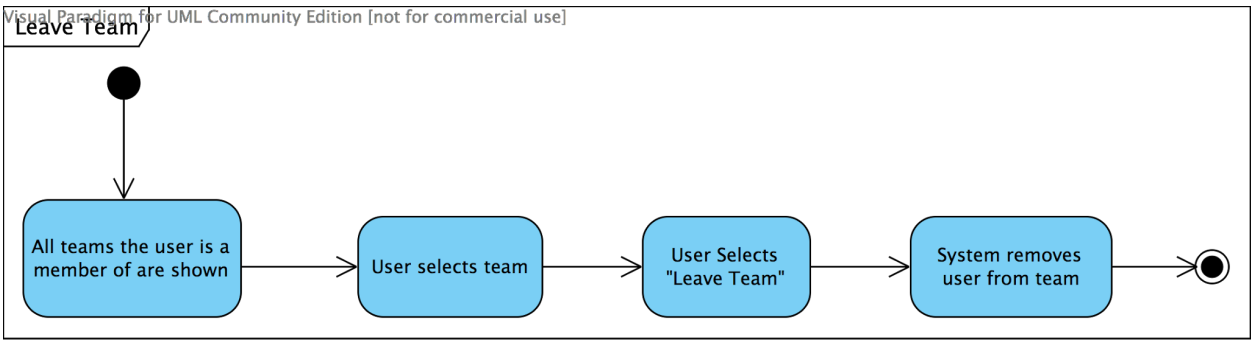

Figure 5: Activity Diagram for "Leave Team" Use Case

- *FR 3.2.9 Leave Team*
- *Description*
  Figure 5 shows how to a player can leave a team. The user selects the team they are a member of and then selects "leave team." The system then removes them from the team.

3.4.6 User Creates Account

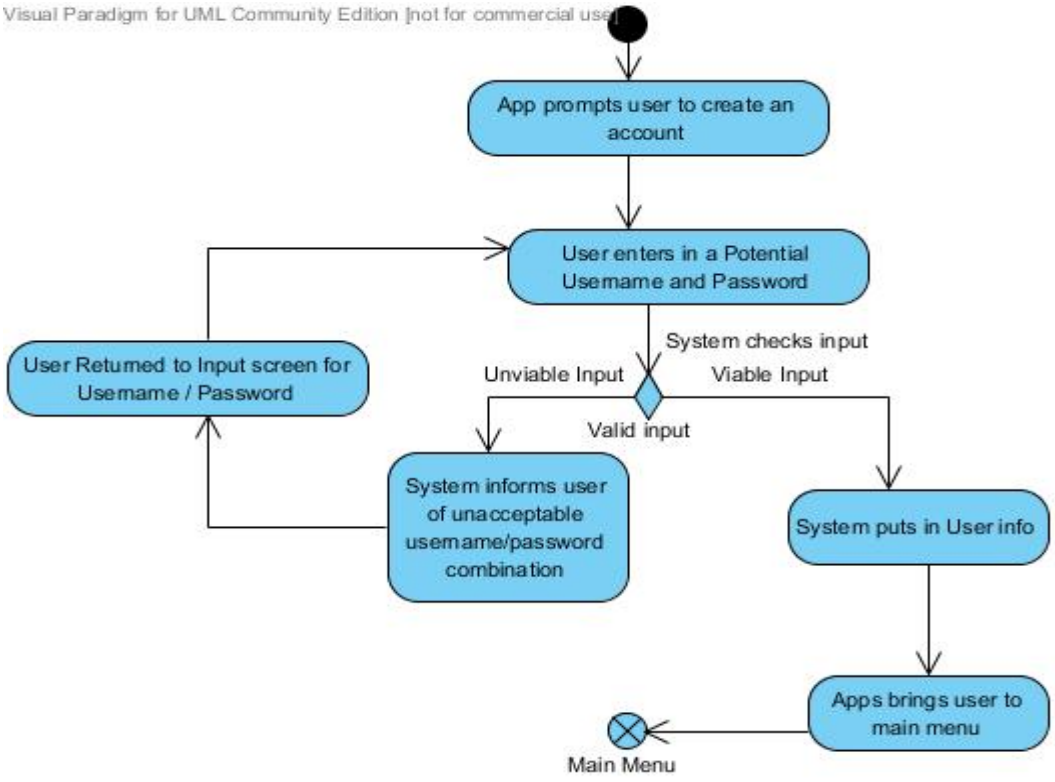

Figure 6: Activity Diagram for "Account Management 3.4.5" Use Case

- *FR 3.2.2 Create Account*
- *Description*

  The diagram shows how a user can create an account. The app prompts a user to create an account, at which point the user enters account information. If the account information is valid (not already in use), the system adds the account to the database and returns the user to the main menu.

3.4.7 Login

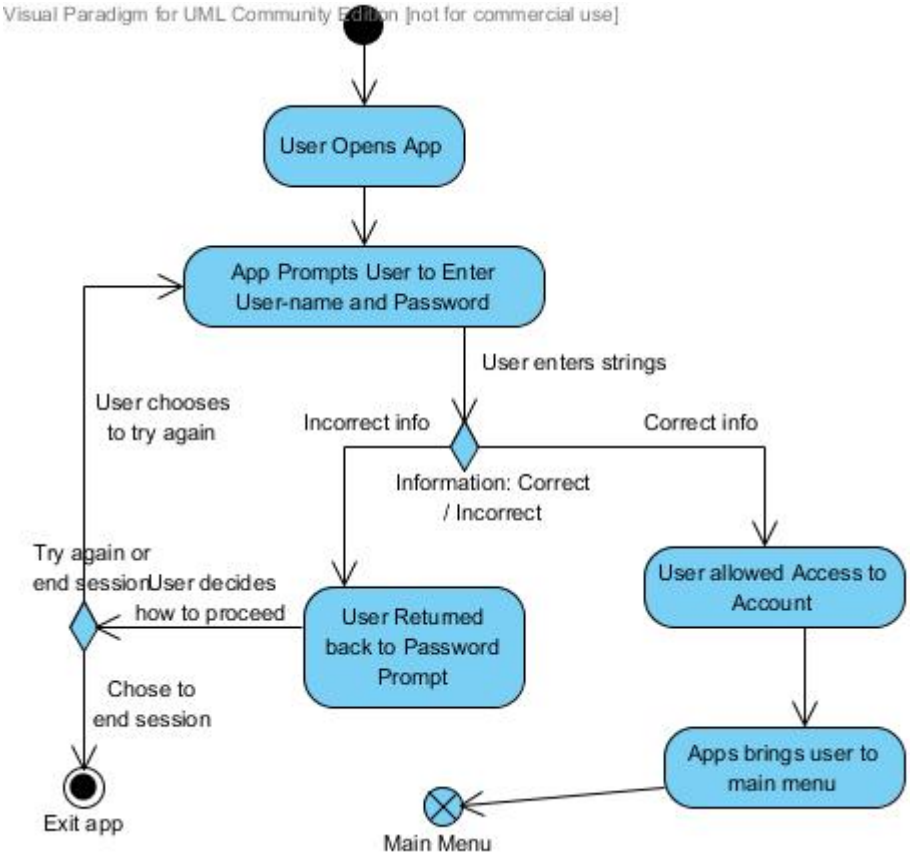

Figure 7: Activity Diagram for "Account Management 3.4.5" Use Case

- *FR 3.2.1 Login*
- *Description*

  When the user opens the app and is not logged in, the user is prompted to login to a user account. The user enters account information and, if the information is correct, the system logs the user in and displays the main menu. If the information is incorrect, the user can re-enter the information or exit the app.

3.4.8 Create or Edit a Play

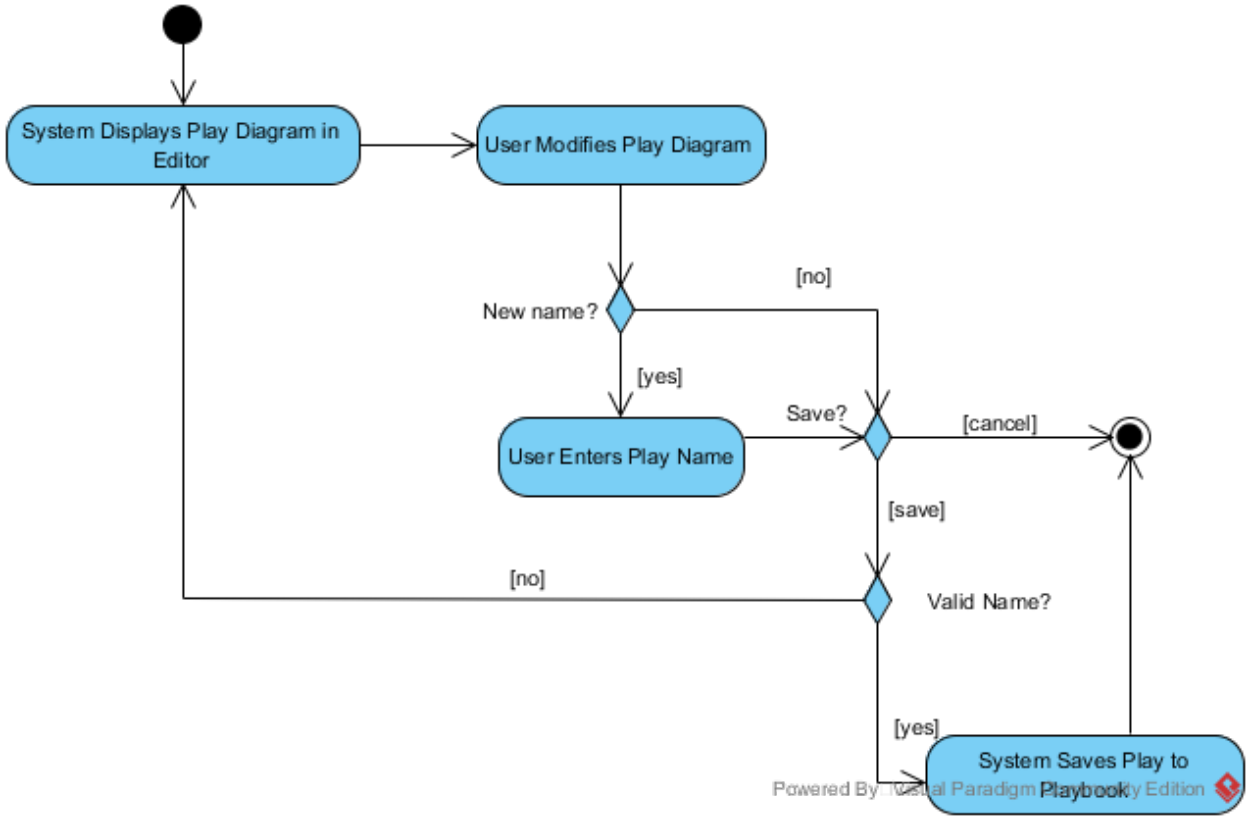

Figure 8: Activity Diagram for "Create/Edit Play" Use Case

- *FR 3.2.12 Create Play, FR 3.2.13 Edit Play*
- *Description*
  The system first displays a play diagram in the play editor. If the user is modifying an existing play, the diagram may already have players and routes in it. Otherwise, the play diagram will be empty. Either way, once the app opens the editor, the user can modify the play diagram as desired, then can optionally enter a new name for the play. After this, the user can either save the new play or cancel the editing session. If the user elects to save the play, and the play name is valid, the play is saved. If the play name is not valid, the user is simply taken back to the editing view.

3.4.9 Manage Playbook

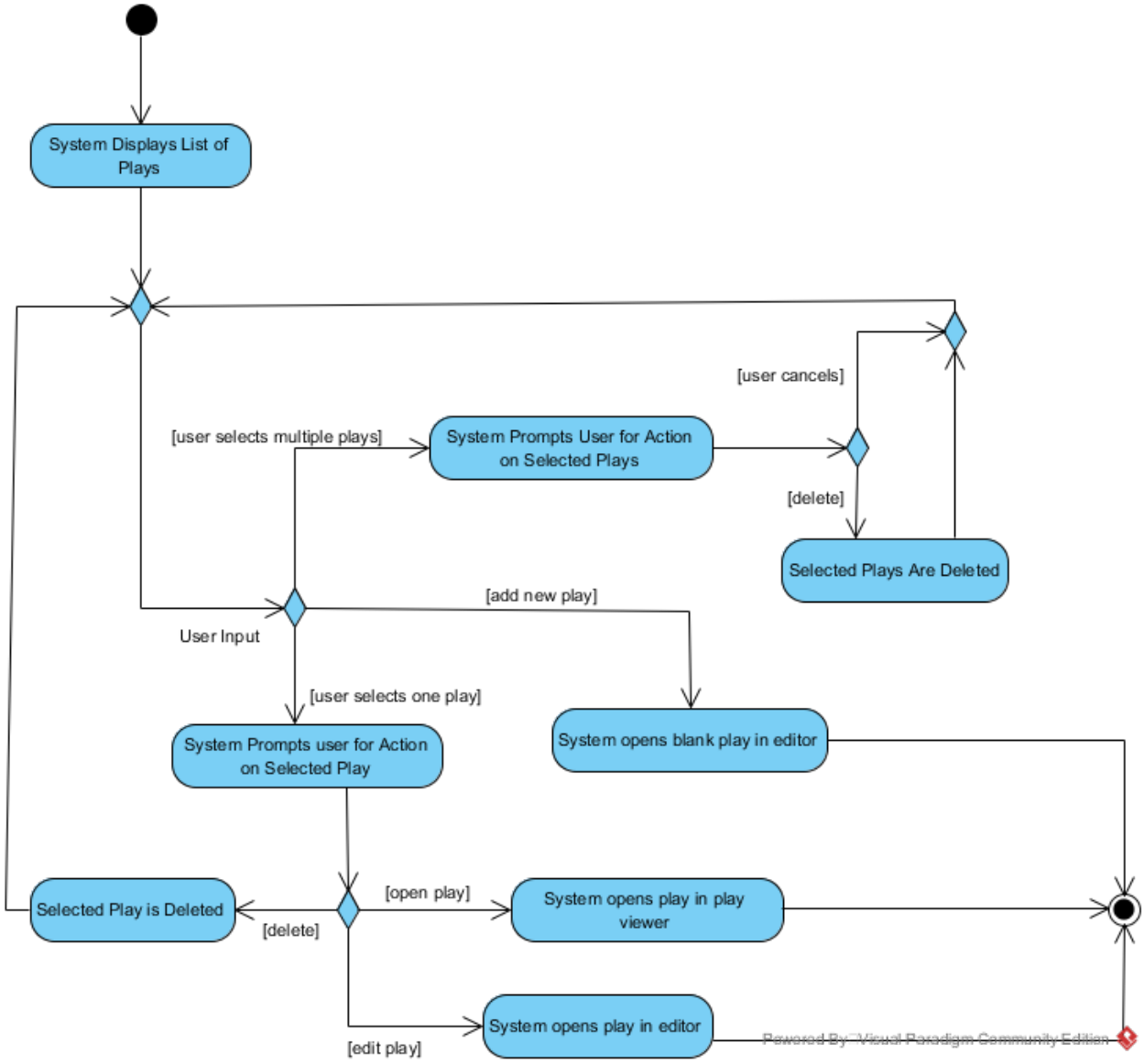

Figure 9: Activity Diagram for "Manage Playbook" Use Case

- *FR 3.2.15 Manage Playbook*
- *Description*
  When the system displays the playbook (or the results of a playbook search), the user has several options for what action to take. The user can select a single play to delete the play, open it in the play view, or open it in the editor. The user can also open up a new play in the editor or select multiple plays to delete at once. If the user deletes plays, the system displays the updated list of plays. If the user opens a play for viewing or editing, the app opens the play in the play view or editor, respectively.

3.4.10 View Play

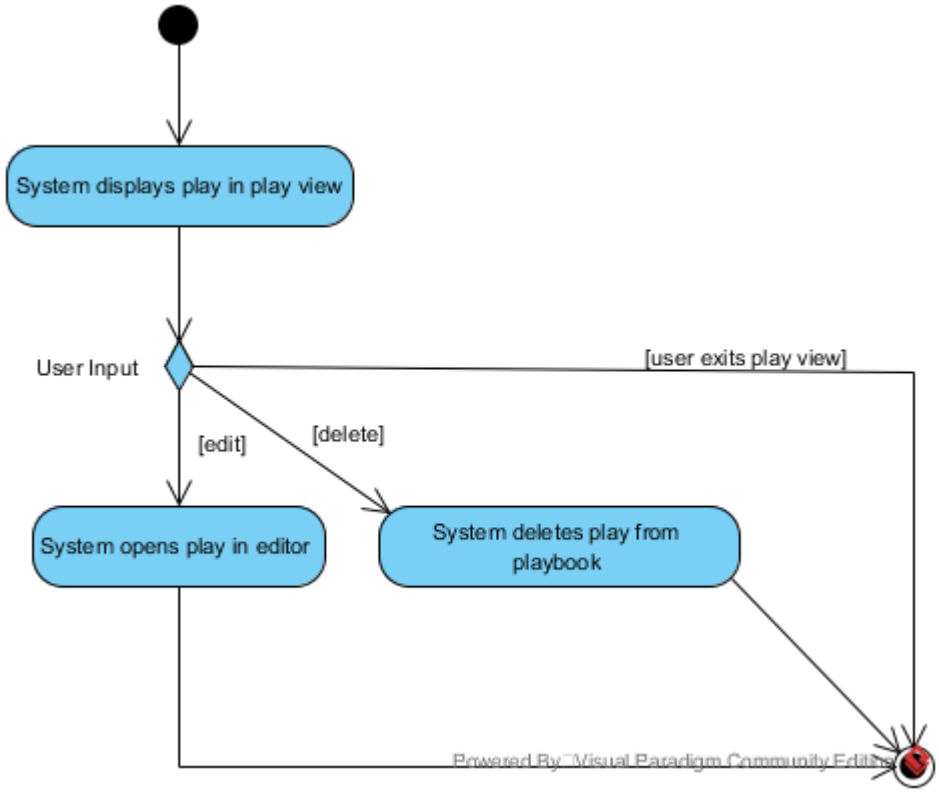

Figure 10: Activity Diagram for "View Play" Use Case

- *FR 3.2.14 Play View*
- *Description*
  The system displays the play view, which shows a single play diagram and allows the user to either open the play in the editor, delete the play, or exit the play view.

3.4.11 Search Playbook

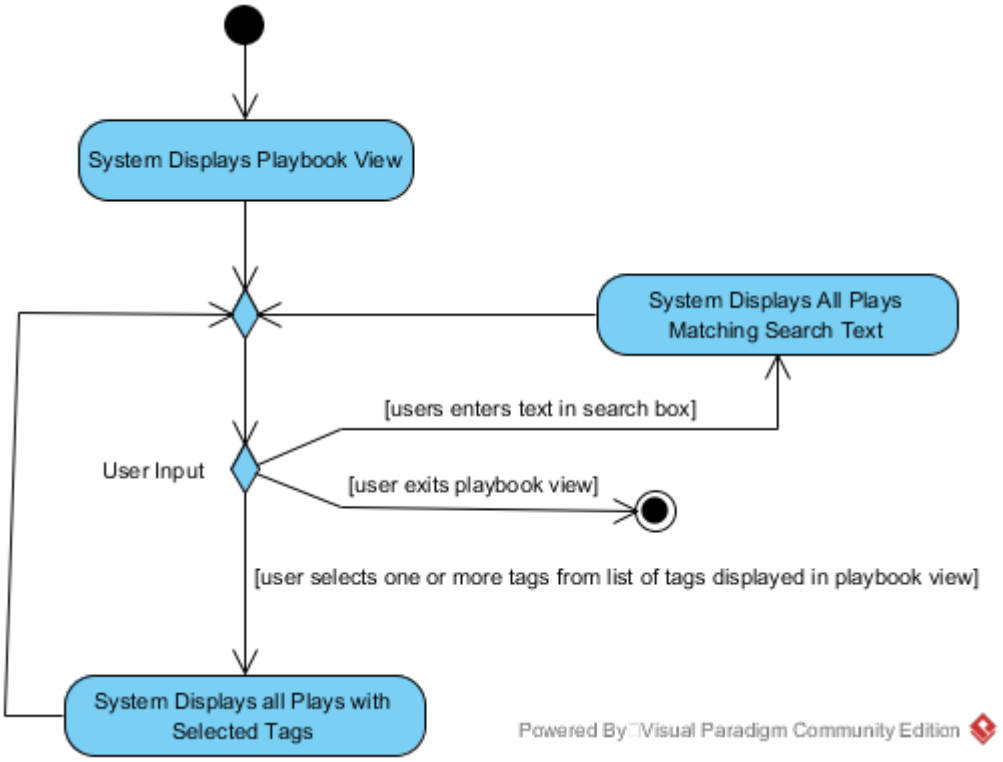

Figure 11: Activity Diagram for "Search Playbook" Use Case

- *FR 3.2.4 Search Playbook*
- *Description*
  When the system displays the playbook view, the user has two options for searching the playbook: entering text in a search bar, or selecting tags from a list of existing tags.

3.4.12 Manage User Account

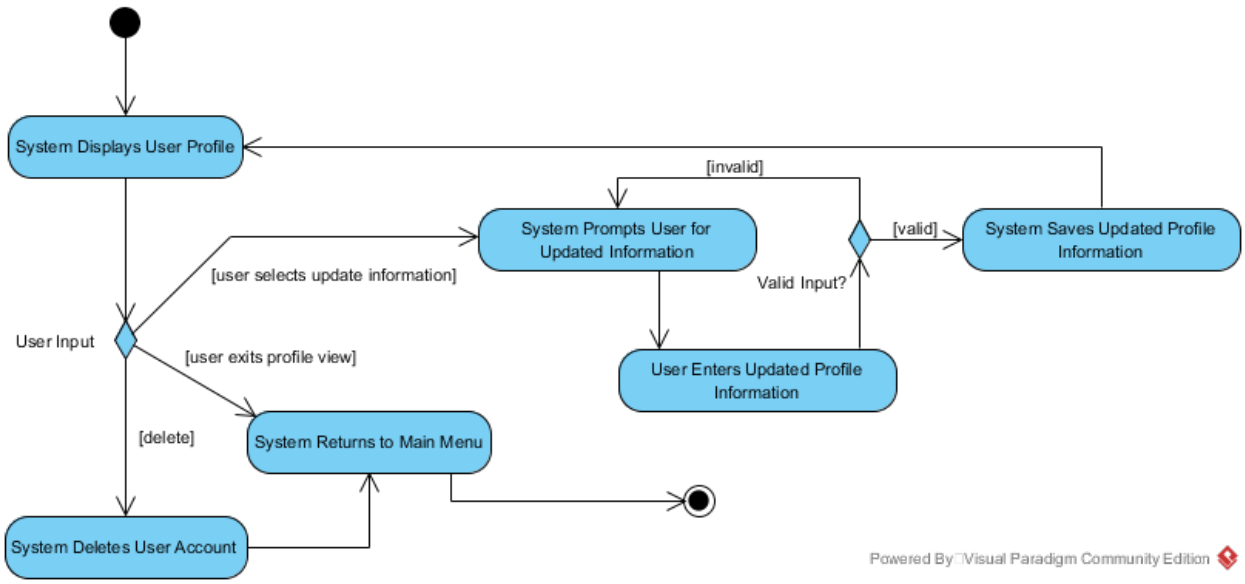

Figure 12: Activity Diagram for "Manage User Account" Use Case

- *FR 3.2.19 Update User Account, FR 3.2.20 Delete User Account, FR 3.2.21 View User Profile*
- *Description*
  When the system displays the user's profile, the user can simply view the profile information and exit the profile view, modify the profile information, or delete the account. If the user chooses to modify the profile and the updated information is valid, the system displays the updated profile. If the user exits the profile view or deletes the profile, the system returns the user to the main menu, deleting the account from the system in the latter case.

3.5 Sequence Diagrams

In this section, sequence diagrams are given which depict class interactions for each activity associated with the app. Throughout this section, diagrams and descriptions refer to ‘views’, which represent Android GUI screens. These views are not modeled in the class diagram and thus do not have specific methods associated with them in the sequence diagrams. This is because all views in the app are assumed to be graphic interfaces only, with all associated methods being built-in Android methods (i.e. onCreate, onResume, etc.). In practice, the views will simply invoke methods within the app’s controller classes, passing in data from user input on screen, rather than processing any of the data themselves.

3.5.1 Create Team

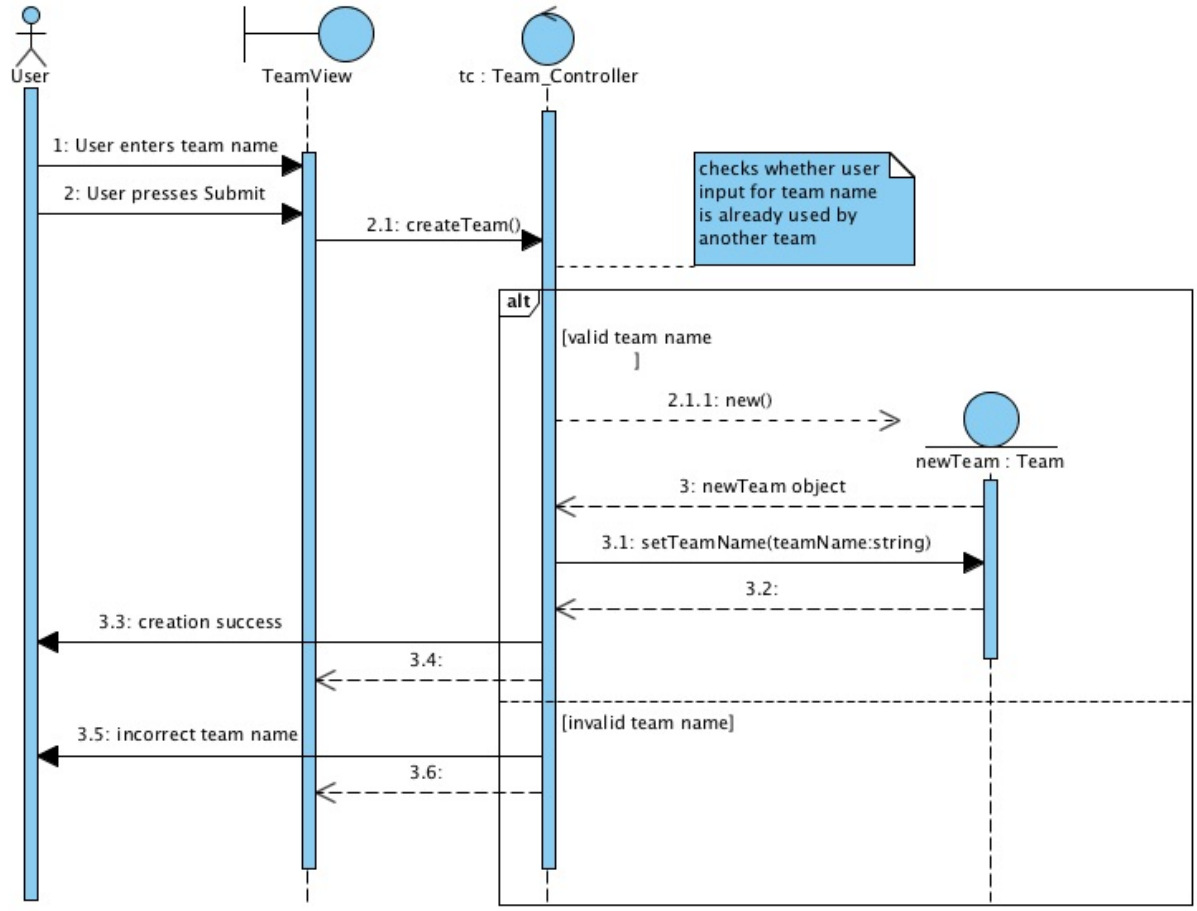

Figure 13: Sequence diagram for “Create Team” use case

- *FR 3.2.6 Create Team*
- *Description*

Figure 13 shows how the system will pass messages in response to a user creating a team. User user will enter a team name into a textField in TeamView and then press submit, which will send a message createTeam() to the Team_Controller. Team_Controller will check whether the team name is valid, and if it is valid, it will create a new Team object and set its name. If the team name is invalid, no team will be created and the user will be notified.

3.5.2 Manage Team

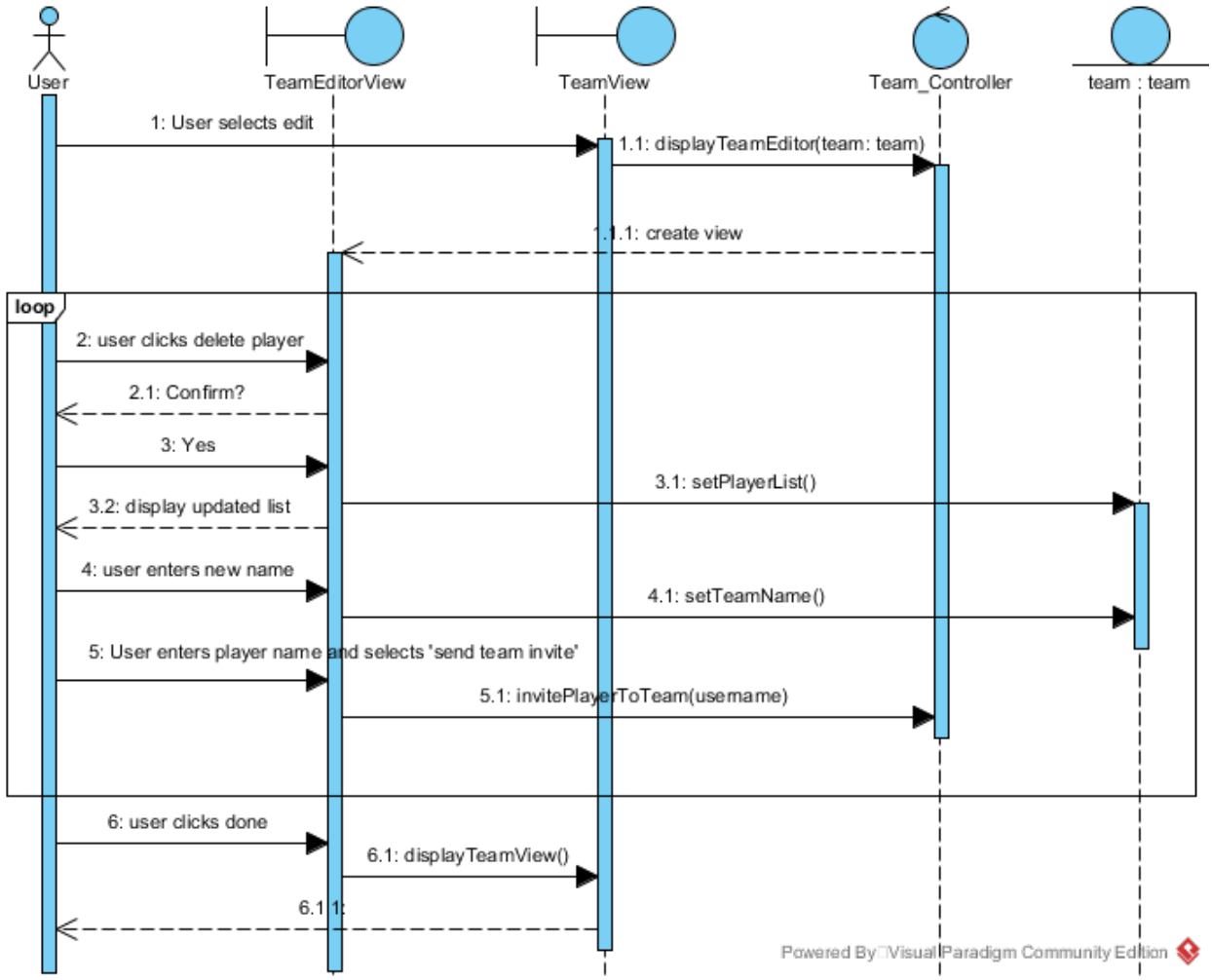

Figure 14: Activity Diagram for "Manage Team"

- *FR 3.2.7 Add Team Member, 3.2.10 Delete Team, 3.2.11 Remove Team Member, FR 3.2.22 User Permissions, FR 3.2.23 Update User Permissions*
- *Description*

  Figure 14 shows class interactions for users managing a team. The diagram is truncated and shows only the interactions once the user is already viewing a team in the Team View. The user selects to enter the 'Editor' mode for a Team, which results in a 'TeamView' being created which passes a message to a 'Team_Controller' that in turn opens a 'TeamEditorView'. From this view, a user

can choose to delete a selected player. They are asked for confirmation and if 'yes' the player is removed from the 'Team' player-list. A user can also choose to add a player, which is done by entering in the player's name. The player's name is then passed in the method 'invitePlayerToTeam' which sends the message to the appropriate player. When user is finished, the user will click 'done' and is returned to the 'TeamView'.

3.5.3 Reply to Team Request

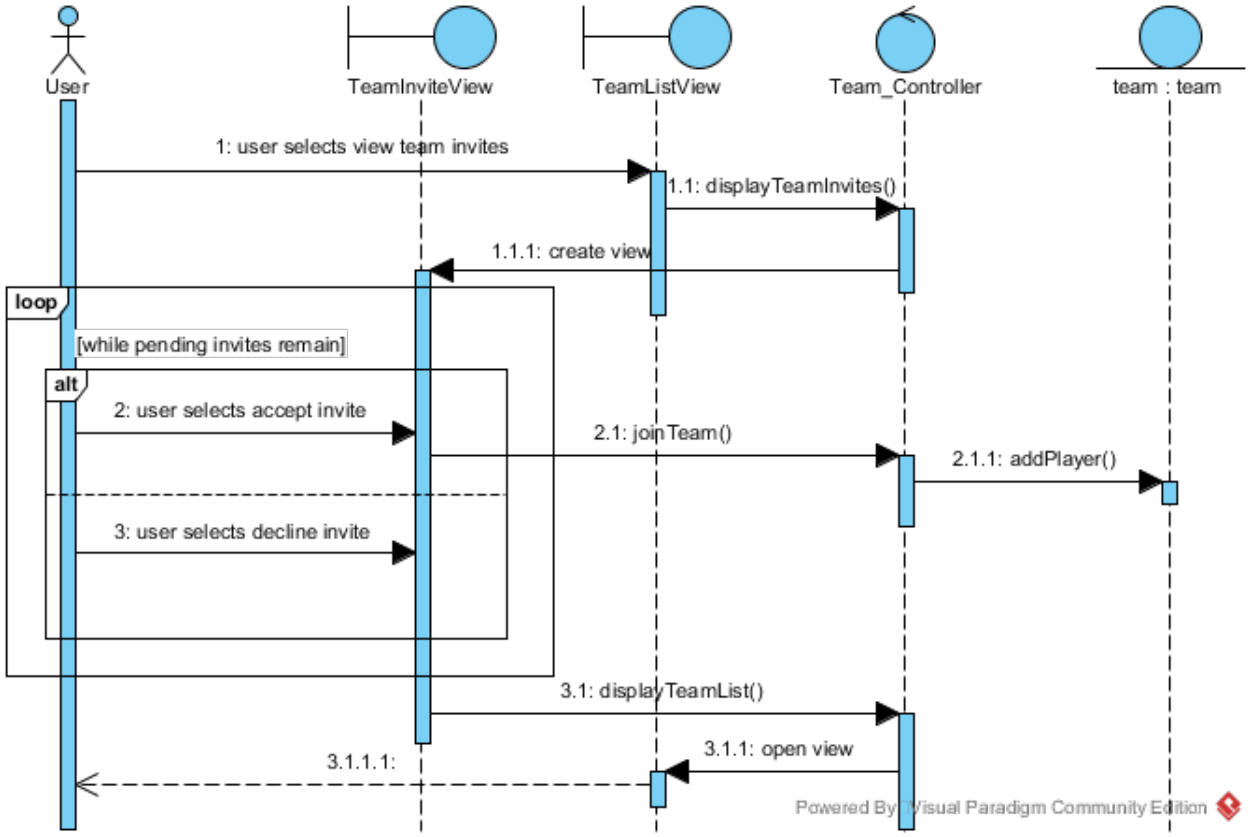

Figure 15: Sequence Diagram for "Reply to Team Request" Use Case

- *FR 3.2.8 Reply to team Request*
- *Description*
  This diagram shows how the system handles a user's reply to a team request. If user accepts, then the system sends the confirmation. Else, the system sends a notice of refusal.

3.5.4 Leave Team

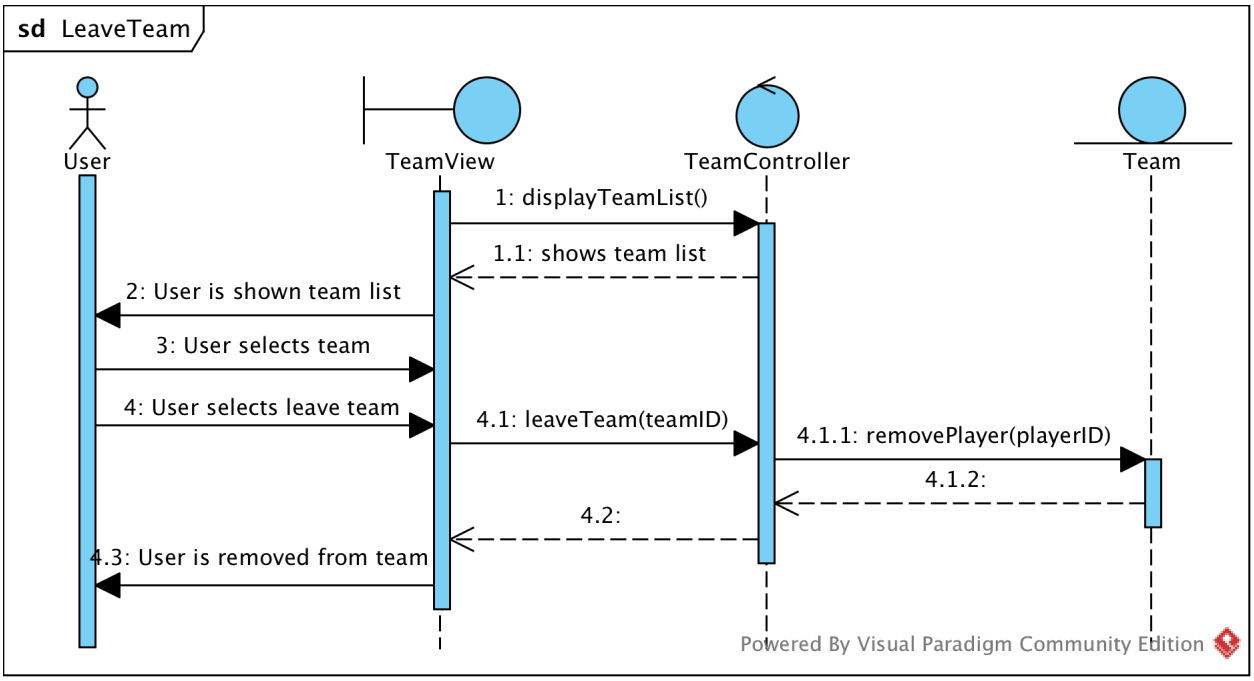

Figure 16: Sequence Diagram for "Leave Team" Use Case

- *FR 3.2.9 Leave Team*
- *Description*
  Figure 16 shows the interaction between classes when the user leaves a team. First, the TeamView (the Android User Interface) sends a message to TeamController to retrieve the teams. TeamController keeps a list of all teams that the user is a member of and returns these to TeamView. TeamView then shows these teams to the user. The user selects a the teams to leave and then selects the "Leave Team," which triggers a call to TeamContoller (leaveTeam()) to remove the user from the selected team. Finally, team controller sends a message to the Team class, which stores data about the team, to remove the user from the team. Once all of this is complete, the user is notified.

3.5.5 User Creates Account

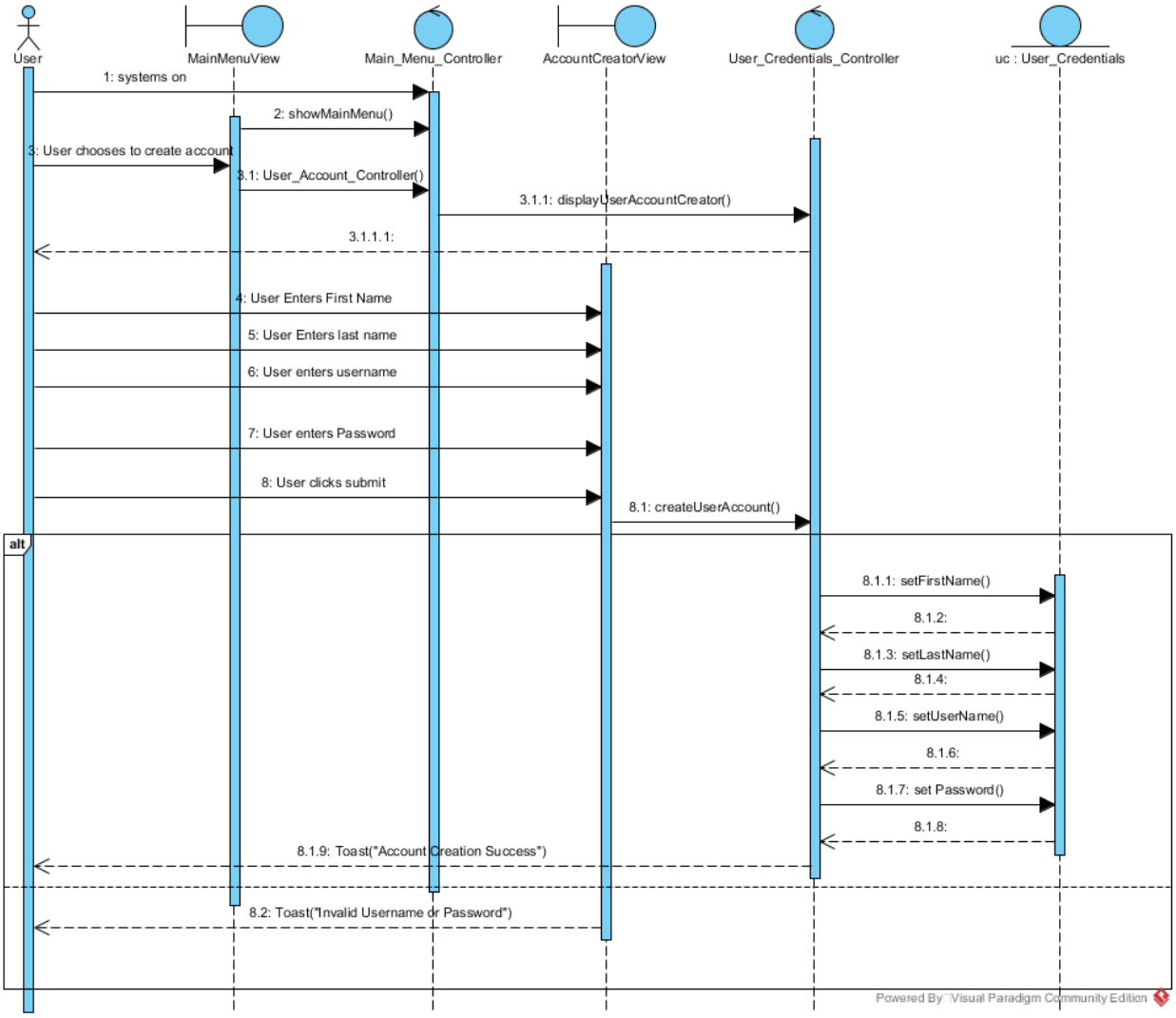

Figure 17: Sequence Diagram for "Account Management 3.4.5" Use Case

- *FR 3.2.2 Create Account*
- *Description*

  The diagram shows how the system responds to a user creating an account. The choice to create an account causes creation of a User_Credentials_Controller that create a new User_Credentails object. Information needed to fill the object is requested from the user and then placed into the object.

3.5.6 Login

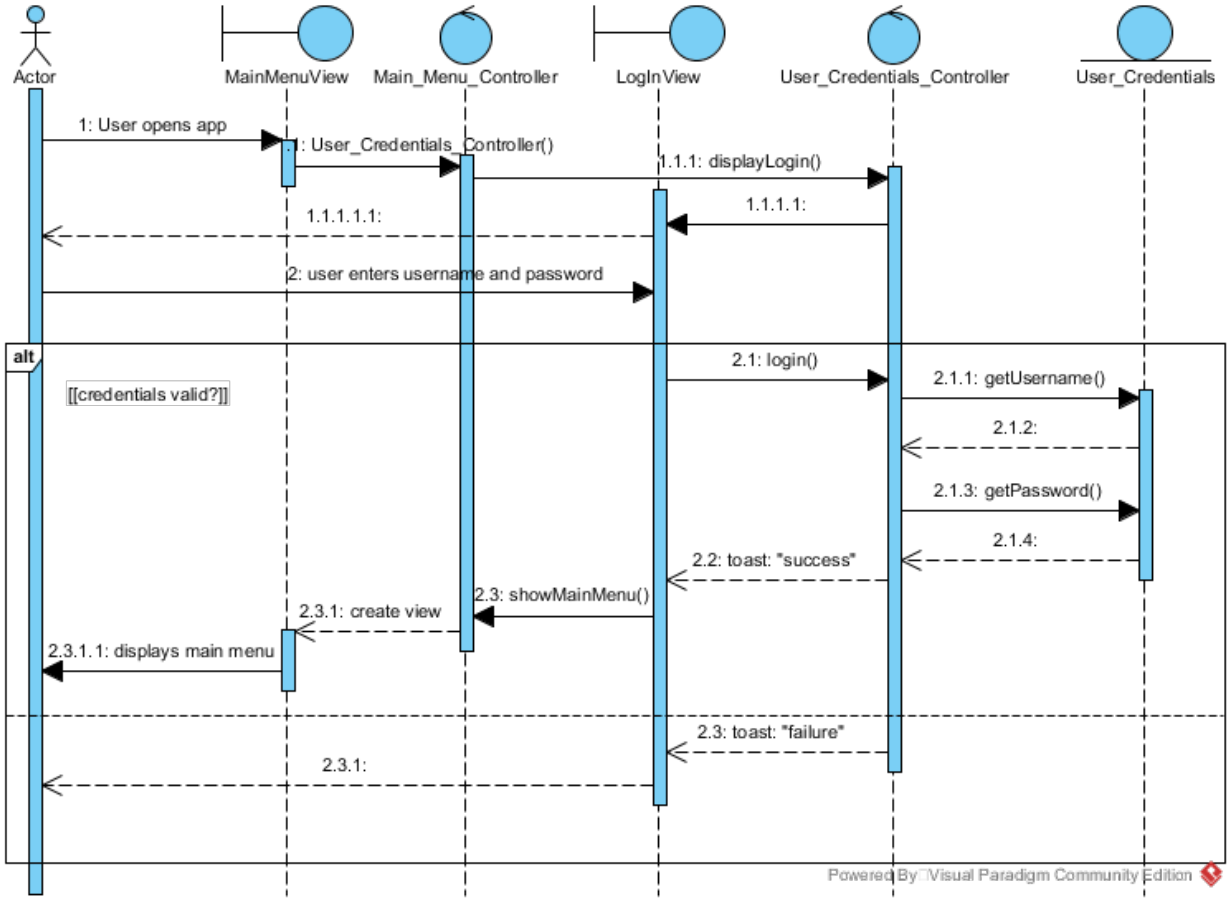

Figure 18: Sequence Diagram for "Account Management 3.4.5" Use Case

- *FR 3.2.1 Login*
- *Description*

  Figure 18 shows the class interactions when a user logs in. The system displays the login UI and then the user enters their credentials. The User_Crededetials_Conroller processes the login credentials and either logs the user in or reports a failure to the user.

3.5.7 Create or Edit a Play

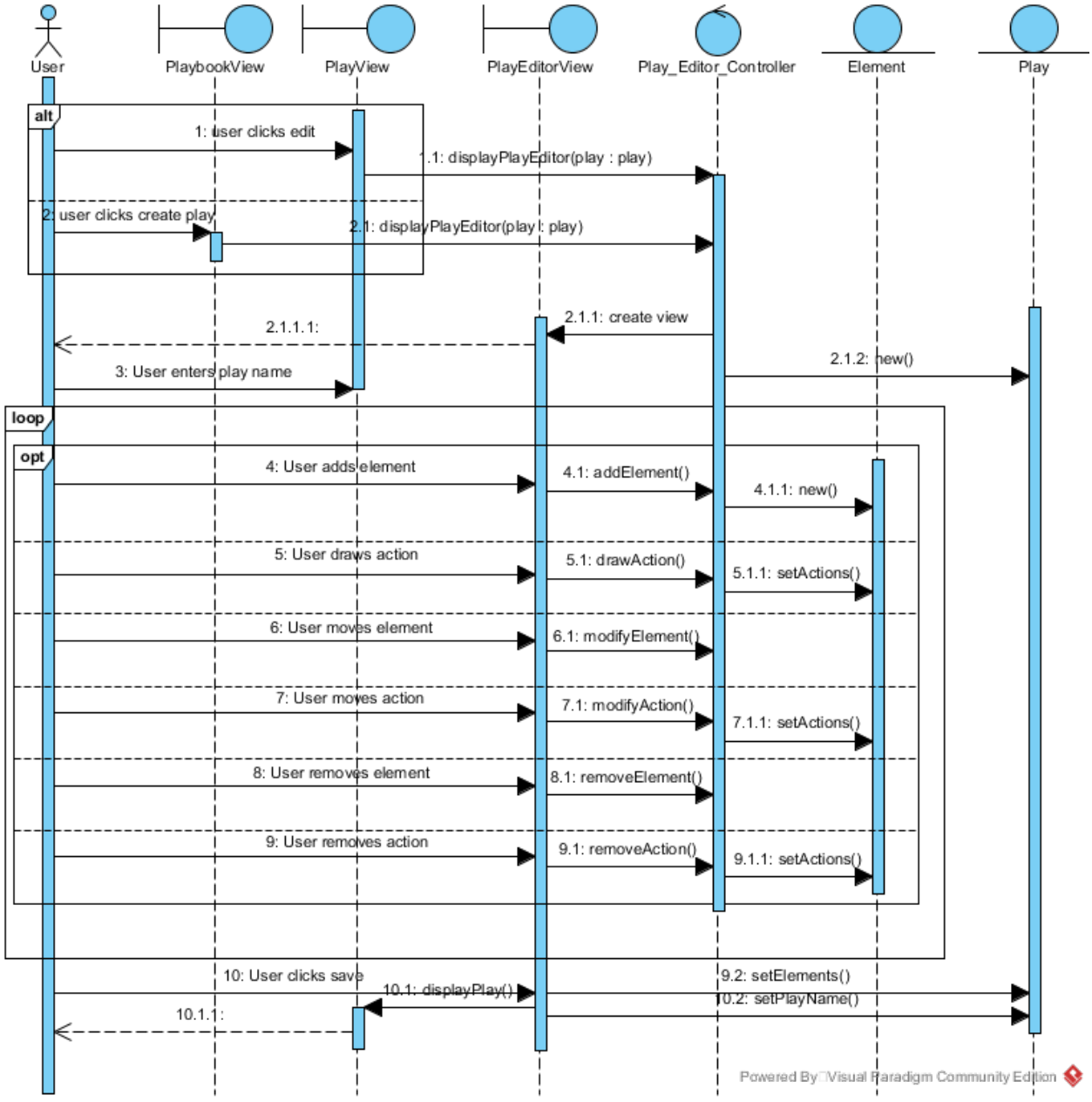

Figure 19: Sequence Diagram for "Create/Edit Play" Use Case

- *FR 3.2.12 Create Play, FR 3.2.13 Edit Play*
- *Description*

The user first indicates that they want to enter the ‘Editor’ mode for a particular play by clicking the button or the user can enter this mode by clicking to create a new play which opens the ‘Editor’ with a blank play. User will have choice of changing (or in the case of a new play changing from a standardized name) the name of the play upon entering the ‘Editor’ mode. In either case an instance of a ‘Play_Editor_Controller’ will be created which will in turn create a ‘Play_Editor_View’. From there the user can repeat as many times as they wish adding, moving, or removing any element or action in the play. Upon exit, the play elements are set and the play name is saved.

a3.5.8 Manage Playbook

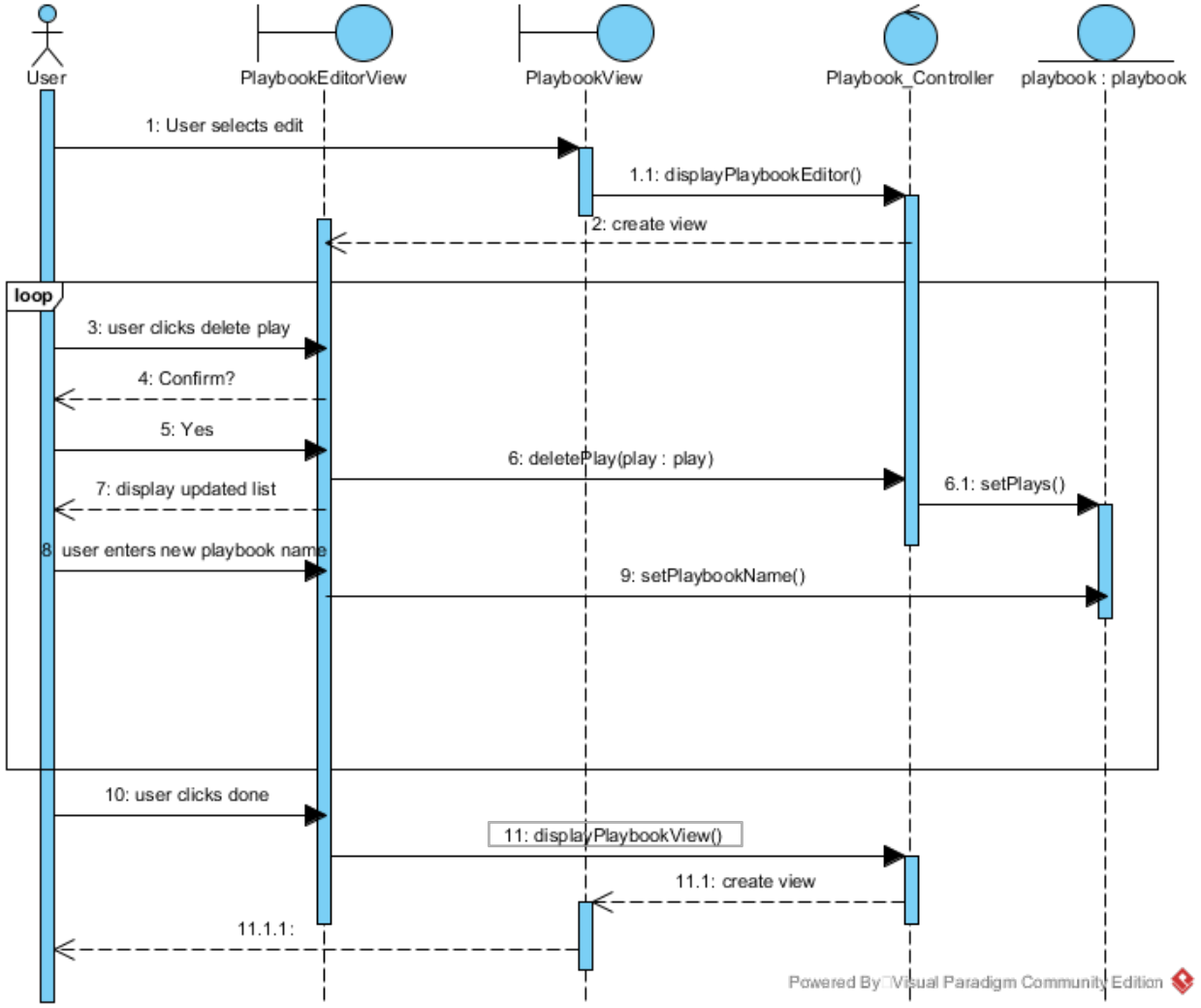

Figure 20: Activity Diagram for "Manage Playbook" Use Case

- *FR 3.2.15 Manage Playbook*
- *Description*
  When the system displays the playbook (or the results of a playbook search), the user has several options for what action to take. The above diagram models the sequence of actions taken when a user wishes to edit the currently displayed list of plays (thus the user is already in the Playbook View).

  When the user clicks 'edit', the playbook view invokes the method to display the playbook editor view within the playbook controller. This method opens the playbook editor view, where the user has the ability to change the name of the

playbook, or delete any number of plays from the currently displayed playlist. Once the user clicks the ‘done editing’ button, the playbook editor view returns the user to the playbook view.

3.5.9 View Play

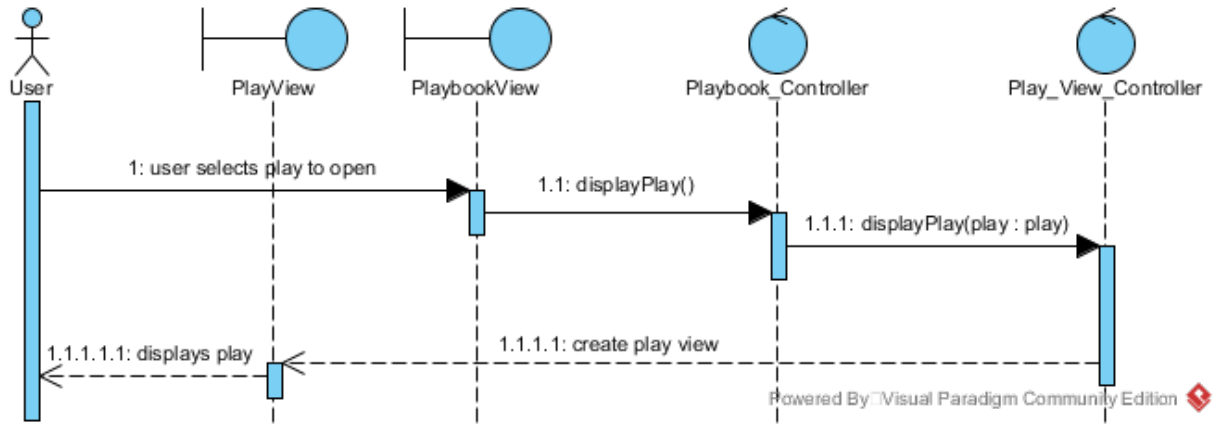

Figure 21: Sequence Diagram for “View Play” Use Case

- *FR 3.2.14 Play View*
- *Description*
  Since displaying a play will occur after searching the playbook, this diagram has been truncated to only show the message passing to display a play. The user selects a play from the displayed list of plays and the system asks user what to do with the play. User selects to view play and the Playbook_Controller creates Play_View_Controller which takes the chosen play and displays it. The user decides to exit viewing play which returns them to the list of plays.

3.5.10 Search Playbook

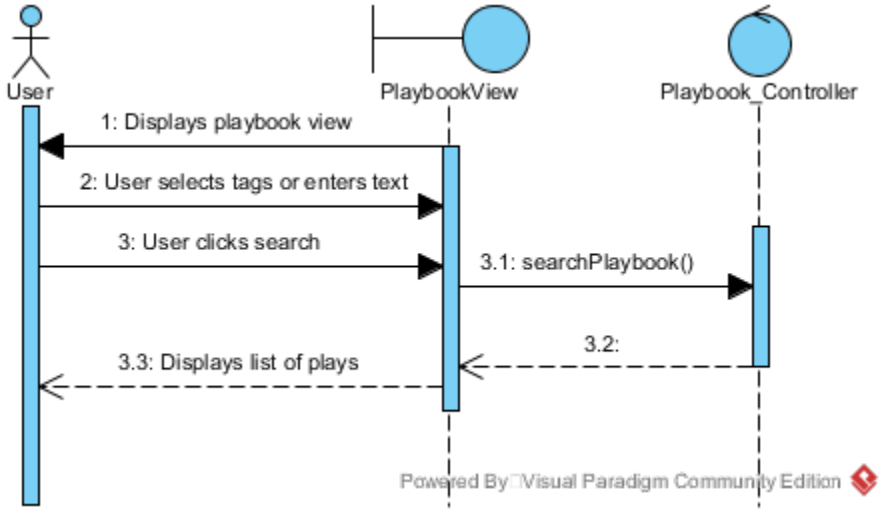

Figure 22: Sequence Diagram for "Search Playbook" Use Case

- *FR 3.2.4 Search Playbook*
- *Description*
  When the user chooses to open playbook, from the main menu, a Playbook_Controller is created. This controller gives user choices of what can be done in the playbook. For search playbook case, the controller allows user to enter search modifiers. The controller creates a Playbook object to interface with back-end playbook data-store and uses the search modifiers to search playbook. Playbook returns the set of plays that meet search criteria. The Playbook_Controller displays the set of plays. Since user could theoretically perform any actions on the set of plays displayed, this diagram has been truncated to only show the search portion.

  To search the playbook, the user either enters text in a search box, or selects a set of pre-defined search tags, then clicks a search button. The playbook view invokes the playbook controller's search method, which filters the plays then updates the list of plays currently displayed by the view to match the results.

3.5.11 Manage User Account

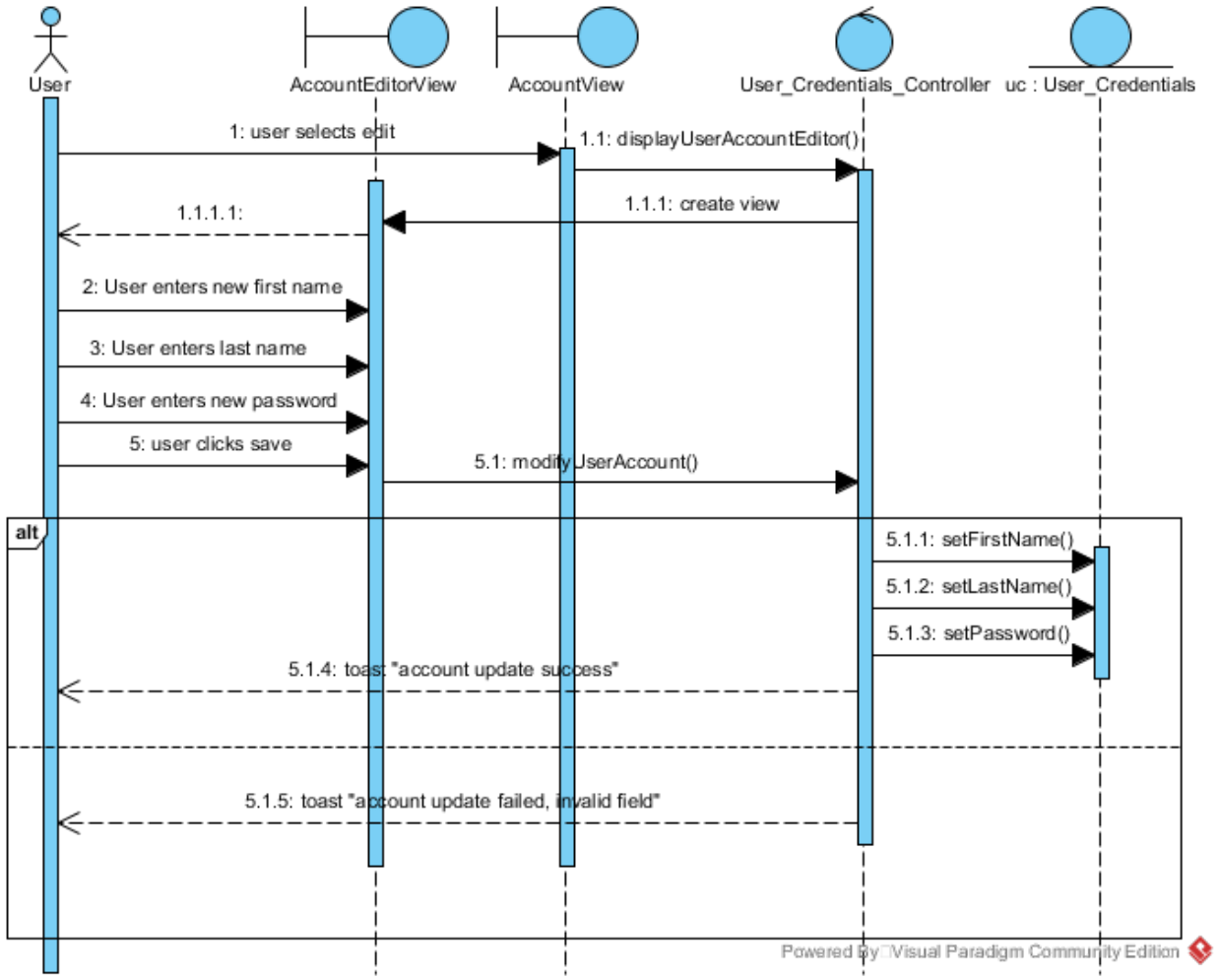

Figure 23: Activity Diagram for "Manage User Account" Use Case

- *FR 3.2.19 Update User Account, FR 3.2.20 Delete User Account, FR 3.2.21 View User Profile*
- *Description*
  Figure 23 shows how the classes interact when the user manages their user account. User_Credentials_Controller creates a view that allows the user to input new data. After the user enters account data (first name, last name, or password), the User_Credentials_Controller modifies a User_Credential object. Then, User_Credentials_Controller alerts the user of success or failure.

3.5 Class Diagrams

The class diagram for the app has been designed based on the MVC (Model, View, Controller) paradigm. This paradigm separates classes into three categories: models, which are simply data objects that store data but do not process it; views, which represent user-facing interfaces; and controllers, which process data from user input and update models.

For each core functionality of the app, there are one or two associated controller classes, as well as one or more model classes. Views, which in the case of the app are Android activities (views/screens) are not modeled in the class diagram. It is assumed that these views will be created graphically and tied directly to the controller classes that call them (i.e., their only methods and attributes will be android-specific traits, e.g. onCreate, onResume, etc.). Thus, any method that begins with 'display' or 'show' calls an android activity and displays a new view accordingly. These views then invoke methods in their corresponding controllers to actually affect underlying data.

3.5.1 High-Level Class Diagram

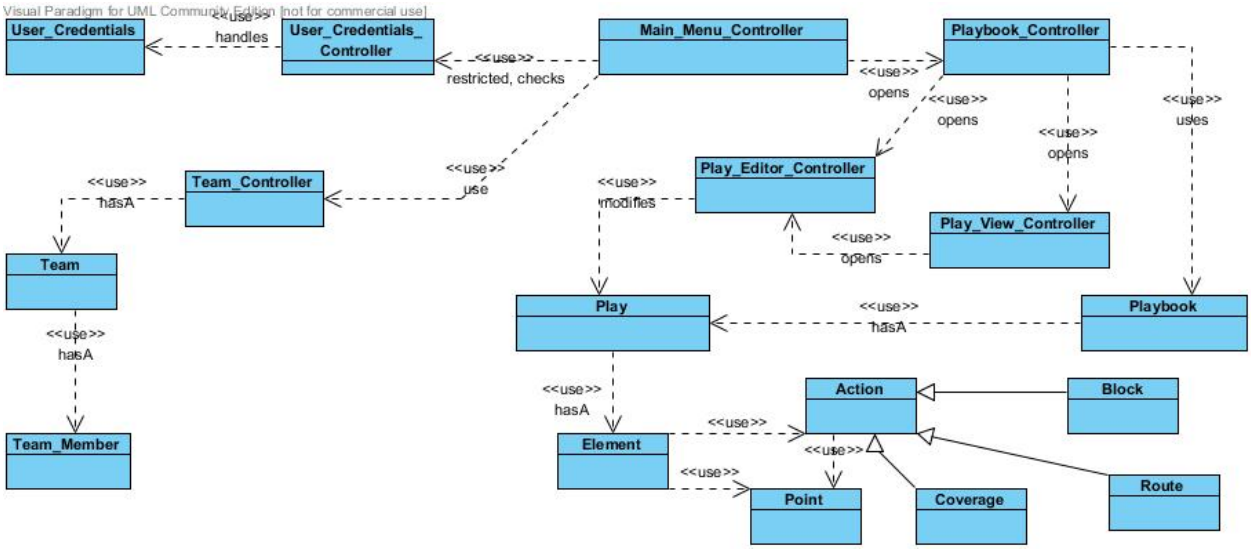

Figure 24: High Level Class Diagram

- *Description*
  Figure 24 shows the high level class diagram for the Playmaker system. It shows all of the classes that will be used to implement the system. The 'Main_Menu_Controller' is the root of the system from which the 'Team_Controller' and 'Playbook_Controller' are called-both of which handle the main functionality of the application. The 'Main_Menu_Controller' also deals with the 'User_Credentials_Controller' which handles user information security. From the 'Playbook_Controller' there are the 'Playbook' which handles the storage of

play-information and the ‘Play_Editor_Controller’/’Play_View_Controller’ both of which will ultimately display a play but under different conditions identified by their names. The ‘Play’ objects consist of ‘Elements’ which occur at some specific ‘Point’ in the diagram. Each element can also be associated with an ‘Action’ appropriately identifying an action that could be described in a football diagram.

3.5.2 Detailed Class Diagram

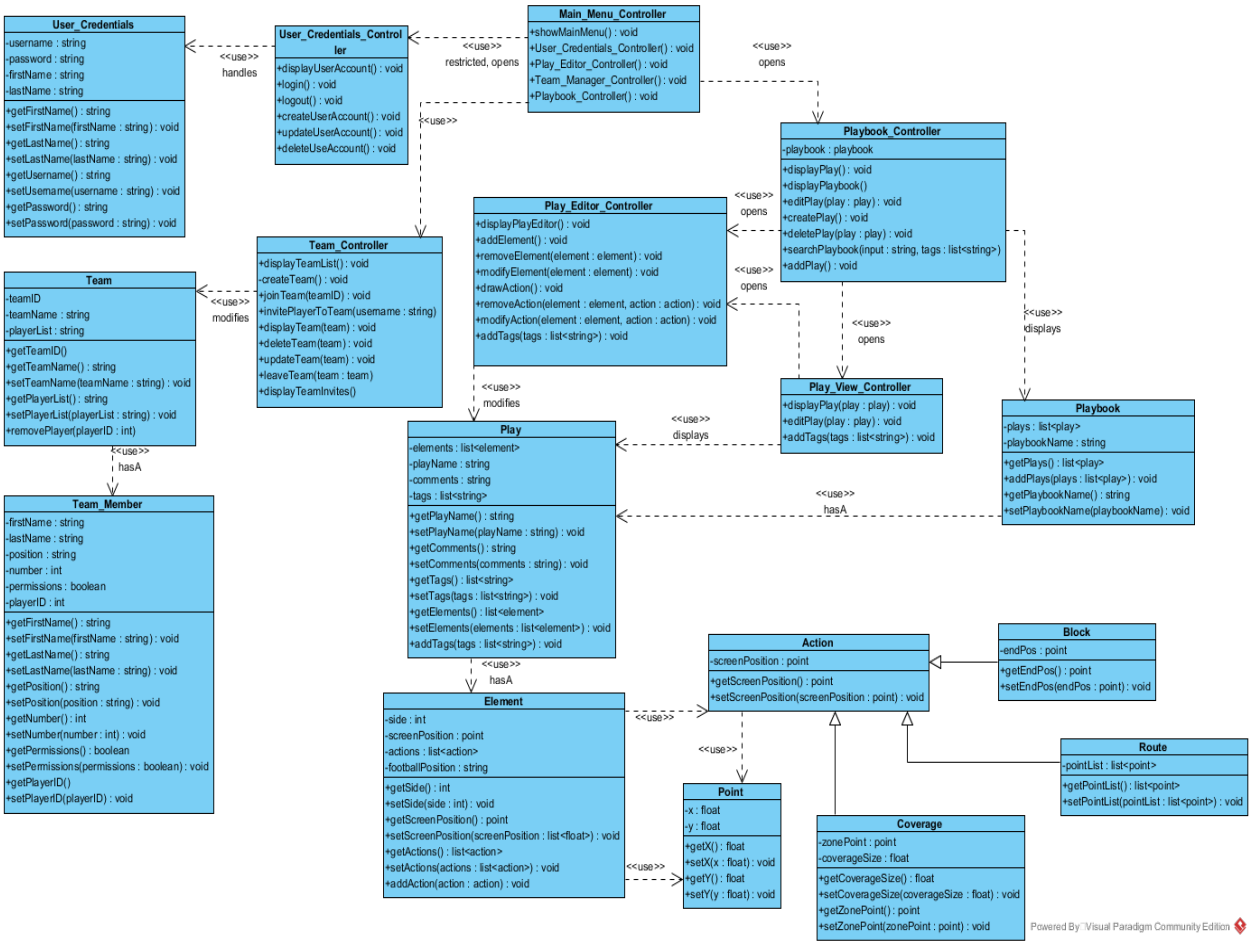

Figure 25: Detailed Class Diagram

- *Description*

  The detailed class diagram above follows the details given in the description of the high-level class diagram (Figure 24). Though not noted above, one can see the inheritance relationship amongst the ‘Action’ class and the ‘Coverage’, ‘Route’, and ‘Block’ classes. Logically an action has both a start point, identified with the ‘Point’ object used by the class and an endpoint which is held by the ‘Action’ object. Example would be a route which would resemble a line. As stated before the ‘Play_View_Controller’ can open a play in the

‘Play_Editor_Controller’ . Important to see here is the inclusion of the ‘setTags’ method of the ‘Play’ class which will form the basis for the search functionality in the playbook as implemented with the ‘Playbook’ class: that being the search for the ‘tags’ associated with a play.

Other controller and model classes function in a similar fashion - each controller has several methods for manipulating instantiations of their associated model classes. For more detailed descriptions of the interactions between the controllers and models, see the sequence diagrams given in the previous section.