Resource Management System

-Documentation-

Project manager: Fangli Andrei
Mentor teacher: Chiorean Dan

Documentation content

Project vision

1) Introduction

Resource Management System is a software build for the staff of an academic institution that are concerned about teaching and most important researching in various domains. The aim of this software is to aid in managing resources (human resources, logistical resources and financial resources) and to decrease the grade of effort required to define various activities.

The software is web based meaning that it can be accessed from the web using any browser from any operating system, th is is useful when researchers are gone on conferences or meetings and are required to add new activities to their projects. They can do that from a device (PC, Mac, tablet etc.) that has a stable Internet connection and one can also view his schedule using the same device. No installation and dependencies are required except a browser (all modern operating systems come with at least one installed browser) and an Internet connection.

2) Positioning

2.2) Problem statement

The director of an academic department manages human resources (hired on different academic or research or administrative positions, having/not having a PhD), financial resources and logistical ones (rooms, equipments, etc). The department is involved in several projects, some of them didactical (undergraduate and master academic programs), some other of a scientific kind (research grants, research cooperations), and some of administrative nature (scientific events, student research groups, admission/finalization exams, business events, etc).

The academic duty of teachers (disciplines, student classes, academic programs, number of hours) is established by a document called "State de functii", produced by using a dedicated software application. The academic schedule is produced each semester by another software system. Both documents are available in electronic form.

A research project has a team of members and several phases, and a phase involves several activities, executed in a specific order. An activity is performed by one or more team members, each with a planned number of hours. The budget of an activity is decomposed into the following expenses: workforce (wages), mobility (covering conference fees, travel, accomodation, and per diem), and logistics (acquisition of consumables and equipments).

An administrative event has a starting and ending date, a list of persons assigned to it and a calendar - list of activities. Each activity consists of several tasks; a task is performed by one or more persons, in a specified time frame (start date, end date), and needs zero/one/more logistic resources.

Our goal is to develop a resource management system (RMS), allowing department's director and members (teachers, research staff, administrative staff) to cooperate and interact for a better use of all department's resources.

The director of the department uses RMS for the definition of research projects and administrative events, approval of some activities and production of reports describing resource allocation and resource use for each department member, each project, each administrative event and each logistical resource.

Academic and research staff uses RMS for getting and recording info related to their activities/tasks, covering didactical, research, and administrative duties.

Administrative staff uses RMS for defining human, financial, and logistic resources of the department.

2.2 Product Position Statement

For	Academic people and Academic Institution Director.
Who	Are concerned about using resources in an optimal manner.
The Resource Management System	Is a software
That	Allows the management of human, logistical, financial resources for various types of activities (research, administrative etc.

3 Stakeholders

3.1 Stakeholder summary

Name	Description	Responsibilities
Administrative staff	Manages the institutions resources (human, logistical, financial).	• Recording human resources • Recording logistical resources • Recording financial resources
Department Director	Creates various activities and approves activity requests from members.	• Defines academic programs • Defines research projects • Defining student research groups • Producing resource usage reports • Approving department activities
Research staff, Teachers	Conduct research and/or didactic activities.	• Performs research activities • Performs didactic activities

3.2) User environment

The users are involved in different activities that may be individual or collective (more than one). For instance a didactic activity is usually performed by one teacher which for research project this is not the case. Research projects are formed of teams and each phases, each phase has a set of activities which can be performed by one or more research project team members.

The duration of a task or activity depends on it's nature, it can take a few hours or a few days or even weeks. Didactic activities usually take up to 2 hours while research projects take a few years.

Some activities are done inside the department institution (courses, seminars etc.) while other activities can take place in various other places (conferences, presentations etc.).

4) Product overview

4.1) Needs and Features

Need	Priority	Features	Planned release
Recording of department resources	1	• Recording of human resources (persons hired on several academic, research, or administrative positions, having/having not a PhD) • Recording of financial resources • Recording of logistical resources (rooms, equipments, etc)	10th April
Defining department activities	1	• Defining of academic programs (undergraduate and master level • Defining research projects (teams, phases, activities) • Defining administrative events (time frame, teams, activities, tasks) • Defining student research groups for specific academic programs • Sending requests for recording new research grants and administrative events (to the director) • Recording of performed research activities (subject to director's approval/confirmation)	24th April
Approving department activities	2	• Approving activities performed by department members	2nd May

		• Approving requests for recording new research grants and administrative events	
Obtaining reports	3	• Producing reports describing resource allocation and resource use for each department member, each project, each administrative event and each logistical resource.	8th May
Accessing information	5	• Getting the calendar of staff activities/tasks (teaching, research, and administrative) • Getting info related to the department's resources • Getting info related to the scientific grants (without financial details and omitting some classified activities, if any) • Getting the calendar of didactical, scientific, and administrative activities (having some filtering features)	22nd May

Supporting Requirements Specification

1) System wide Functional Requirements

The product is a software web based application that requires a server as a host for the application and a browser to use it. In order to install the application an IIS (Internet Information Services) instance must be installed on the targeted server and a SQL Server instance must also be installed (can be on a remote server or the same with the one where the application is hosted).

2) System Qualities

2.1) Usability

The product is easy to use because it requires only a browser to run and has a friendly user interface. One without much knowledge in computer science can easily benefit from all features that this product offers.

2.2) Reliability

The product can be ran from multiple browser at the same time and perform different tasks. Each user runs on a different session on the server meaning that any user, without depending on how many users are using the application at the same time, cannot interfere with the work of another user. Each user is treated independently.

2.3) Performance

The product is very responsive and does not require any start time or shutdown time since the application is hosted on a server. When a user accesses the application through the browser (as a web site) the application is already loaded and waiting for users. Also, there is no shutdown time required. The amount of time the user has to wait is exactly the same as the amount of time the respective user needs to close his/her browser.

2.4) Supportability

Being a web application it is accessible from anywhere through the web (Internet) using a browser. Any user can benefit from all features that this product provides from any operating system that has a browser. No extra installations are required and no dependencies are used.

3) System Interfaces

3.1) User Interface

3.1.1) Look & Feel

The application uses a rich graphical user interface that is very friendly. All content that is displayed automatically re sizes when the user resizes his browser. The content is made to fit the browser display area allowing the user to take advantage of any displays he has at hand. It may be a tablet, a PC or a TV. No matter on the size, the content will automatically resize and fit any given display area.

3.1.2) Layout and navigation

The product offers the user a simple, yet effective layout that features a header, a menu on the left side and a content area that takes advantage of most of the display area. The navigation is made simple through menus and sub-menus, this allows the user easily find and use a feature.

3.1.3) Consistency

The user interface is consistent. Some features take the user through a set of steps, each step allows the user to decide what will happen next and also present him the data he already gave allowing him to focus more on providing the correct information rather than focusing on remembering what he did three steps earlier.

4) System constraints

The application requires a server as a web host and a SQL Server instance. The web server should have at least a quad core CPU and at least 8 Giga Bytes of RAM and around 4 drivers in RAID-10 to cover I/O with external devices such as hard disks. Most web hosts can provide such hardware.

A DNS server is recommended to help users remember a domain (such as rms.myDomain.com) rather than a IP address (e.g.: 132.42.231.2). Domains are more user friendly and in case a few letters were forgotten, a simple search on the web can find the domain, however when dealing with IP addresses directly, online web search can't do much if one forgets a digit in the address.

Use Case Specifications

1) Check User Identity

Description: the user wants to authenticate in order to use some features.

Preconditions: none.

Postconditions: if the login succeeds the user can make use of all features he is allowed to use.

1.1) Main flow:

1. The user selects the option to authenticate
2. The system displays a page with login fields
3. The user types his login details
4. The user clicks on authenticate
5. The system authenticates the user and shows him the main page

1.2) Alternative flow:

1.2.1) Invalid login

At step 5, if the login fails then the system displays an error message.

2) Register Human Resource

Description: An administrative person adds a new human resource (teacher, student etc.).

Preconditions: The administrative person must be authenticated.

Postconditions: The human resource is added in the system.

2.1) Main flow:

1. The administrative person selects the option to add a new human resource from the main panel
2. The system asks the administrative person to complete a form
3. The system checks and confirms the data for the correct format
4. The system displays an "operation successful" message and asks the administrative person if he wants to add a new human resource or to return him to the main panel

2.2) Alternative flows:

2.2.1) Invalid data format

Description: The administrative person enters incorrect data (e.g: digits in the name field).

1. The system notices the error (at step 3)
2. The system displays a message containing the field name with invalid data and the requirements of acceptance for the respective field
3. The system returns the administrative person to the form leaving all fields unchanged2.2.2) Unfilled fields

Summary: **t**he administrative person leaves some fields unfilled. The system checks if all fileds are filled (at step 3).

1. The system notices that there are uncompleted fields
2. The system displays a message containg the names of all fields that are not filled
3. The system returns the administrative person to the form leaving all fields unchanged

2.2.3) Human resource already exists

Summary: the system validates the submitted form and checks if the data is not already present in the system.

1. The system notices that the given data is already registered
2. The system displays a message containing the reason why the human resource cannot be added (duplicates are not allowed)
3. The system returns the administrative person to the form leaving all fields unchanged

3) Register financial resource

Preconditions: the administrator must be authenticated to be able to add a financial resource

Description: an administrative person wishes to register a financial resource.

Postconditions: if the provided information is corret, the financial resource is registered, otherwise the system database remains unchanged.

3.1) Main flow:

1. The administrator selects the option to add a financial resource
2. The system displays a form for the admin to complete (value, currency etc.)
3. The administrator completes the form
4. The administrator clicks on the register resource button
5. The system registers the financial resource into its database

3.2) Alternative flows

3.2.1) Unfilled fields

Summary: At step 4, at least one field is left unfilled then the system allerts the user about the mistake

3.2.2) Invalid data format

Summary: At step 4, some fields are left unfilled then the system allerts the user about the mistake

4) Register logistical resource

Description: an adminsitrator wants to add a logistical resource (class room, equipment etc.)

Preconditions: the administrator must be authenticated.

Postconditions: if all data is valid the logisitical resource is registered, otherwise the system databse is left unchanged.

4.1) Main flow:

1. The admin selects the option to add a logistical resource
2. The admin selects the category into which he wishes to add the logistical resource (class room, equipment etc.)
3. The system displays a form for the corresponding logistical resource category
4. The admin fills the form
5. The admin submits the form to the system
6. The system notifies the admin with a success-like message

4.2) Alternative flows:

4.2.1) Unfilled fields

At step 5 there are fields left empty then the system notifies the user about the mistake.

4.2.2) Invalid data format

At step 5 there are fileds with incorrect format then the system notifies the user about the mistake.

4.2.3) Logistical resource already existent

At step 5 the logistical resource is already present in the system database then the system notifies the user about the mistake.

5) Import “stat de functii”

Description: an administrator wants to import the “Stat de functii” from a document generated by an external software, containing the duty of teachers.

Preconditions: The administrator must be authenticated. The document “Stat de functii” must be generated by an external application in a electronic format. Example html document.

Postconditions: the document “Stat de functii” is imported in the system. If an error occurs, no data is changed. If the software is able to load the document again, it will rewrite everything.

5.1) Main flow

1. The administrative person selects the import “Stat de functii” feature
2. The system asks for the location of the document
3. The administrator selects the document location
4. The system imports the selected document
5. The system confirms that the operation succeeded.

5.2) Alternative Flows

5.2.1) Unselected file

At step 4 there is no document selected then the system notifies the user about the mistake.

5.2.2) Invalid data format

At step 4 the selected document is not a “stat de functii” document then the system notifies the user about the mistake.

6) Import schedule

Description: an administrator wants to import the schedule from a document generated by an external software.

Preconditions: the administrator must be authenticated. The document that contain the schedule must be generated by an external application in an electronic format. Example html document.

Postconditions: the document that contain the schedule is imported in the system. If an error occurs, no data is changed. If the software is able to load the document again, the system will rewrite everything.

6.1) Main flow

1. The administrative person selects the import schedule feature
2. The system ask for the location of the document
3. The administrator selects the document location
4. The system imports the schedule from the selected document
5. The system confirms that the operation succeeded

6.2) Alternative flows

6.2.1) Unselected file

At step 4 there is no document selected then the system notifies the user about the mistake.

6.2.2) Invalid data format

At step 4 the selected document is not a “schedule” document then the system notifies the user about the mistake.

7) Define study program

Description: the director wants to define a new study program.

Preconditions: the director must be authenticated.

Postconditions: if the data is correct, then the application updates its database, otherwise no changes are made.

7.1) Main flow

1. The director selects the define study program option
2. The system displays a form for the director to complete
3. The director completes the form and submits the form
4. The system confirms that the operation succeeded

7.2) Alternative flows

7.2.1) Unfilled fileds

At step 3 there are unfilled forms then the system notifies the director about the mistake.

7.2.2) Invalid data format

At step 3 some fields contain invalid data format (instead of number of years letter ‘a’ is inserted).

7.2.3) Study program already existent

At step 4 the system notices that a study program for the targeted domain already exists then it allerts the user about the situation.

8) Define research projects

Description: the director wants to add a research project.

Preconditions: the director must be authenticated.

Postconditions: if the data is valid, the system will be updated. Otherwise no data is changed.

8.1) Main flow

1. The director selects the option to create a new research project
2. The system displays a form for the director to complete
3. The director completes the form and submits it
4. The system confirms that the form data is valid
5. The system displays a form for adding phases to the research project
6. The director completes the forms and submits
7. The system confirms that the form data is valid
8. The system adds the phases to the research project
9. The system displays, for each phase, forms for adding research activities
10. The director completes the forms and submits
11. The system confirms that the data is valid
12. The system creates the research project with all the provided data

8.2) Alternative flows

8.2.1) Unfilled fields

At 3, 6 or 10 the director does not complete at least one of the fields then the system displays an allert message and returns the director to the previous step.

8.2.2) Invalid data format

At 3, 6 or 10 the director does not complete a filed accordingly (instead of a date he enters the letter ‘a’) then the system displays an allert message and returns the director to the previous step.

9) Define student research group

Description: the director wants to define a student research group.

Preconditions: the director must be authenticated.

Postconditions: if the data is valid the system databse is updated, otherwise no changes are made.

9.1) Main flow

1. The director selects the option to add a new student research group
2. The system displays a form for the director to complete
3. The director completes the form and submits
4. The system confirms that the operation succeeded

9.2) Alternative flows

9.2.1) Unfilled fields

At 3, the director does not complete at least one of the fields then the system notifies the director about the mistake.

9.2.2) Invalid data format

At 3, the director does not complete the fields accordingly (instead of a number he types the letter 'a') then the system notifies the director about the mistake.

10) Define administrative event

Description: the director wants to create a new administrative event.

Preconditions: the director must be authenticated.

Postconditions: if all given data is valid, the system database is updated otherwise no data is changed.

10.1) Main flow

1. The director selects the option to create a new administrative event
2. The system displays a form for the director to complete
3. The director completes the form and submits
4. The system confirms that the data is valid
5. The system displays forms for creating teams for the administrative event
6. The director completes the forms and submits
7. The system confirms that the data is valid
8. The system displays forms for creating activities for the administrative event
9. The director completes the forms and submits
10. The system confirms that the data is valid
11. The system displays forms for adding tasks for each activity
12. The director completes the forms and submits
13. The system confirms that the data is valid
14. The system confirms that the operation succeeded

10.2) Alternative flows

10.2.1) Unfilled fields

At 3, 6 or 10 the director does not complete at least one of the fields then the system displays an allert message and returns the director to the previous step.

10.2.2) Invalid data format

At 3, 6 or 10 the director does not complete a filed accordingly (instead of a letter he enters the letter 'b') then the system displays an allert message and returns the director to the previous step.

11) Obtain resource usage reports

Description: the department director wants to obtain usage reports for all resources.

Preconditions: the director must be authenticated.

Postconditions: the system provides the resource usage reports.

11.1) Main flow

1. The director select the option to receive resource usage reports
2. The system provides the report

11.2) Alternative flows

11.2.1) No resource usage

If there is no resource usage the system displays a message telling that no resource has been used.

12) Approves member activities

Description: the director wants to approve activities proposed by members.

Preconditions: the director must be authenticated.

Postconditions: the system updates activities.

12.1) Main flow

1. The director selects the option to approve activities proposed by members
2. The system displays the pending activities
3. The director selects which activities to approve
4. The system updates the selected activities

12.2) Alternative flows

12.2.1) No pending activities

If there are no pending activities a message is shown telling that there are no pending activities.

13) Get attending activities

Description: a teacher wants to see the calendar of activities he is attending.

Preconditions: the teacher must be authenticated.

Postconditions: the system displays the requested information.

13.1) Main flow

The teacher selects the option to get a calendar of all activities he is attending

The system displays the requested information.

13.2) Alternative flows

13.2.1) No activities

If the teacher is not attending any activities the system displays a message that informs him of the situation.

14) Request activities

Description: a teacher wants to request activties from the director.

Preconditions: the teacher must be authenticated.

Postconditions: the system submits an activity request if all data is valid, otherwise no request is sent.

14.1) Main flow

1. The teacher selects the option to request activities from the director
2. The system displays a form for the teacher to fill
3. The teacher fills the form and submits
4. The system confirms that the operation succeeded

14.2) Alternative flows

14.2.1) Unfilled fields

At 3, the teacher does not fill at least one of the fields then the system displays an allert telling the teacher about the situation.

14.2.2) Invalid data format

At 3, the teacher does not complete at least one of the fields accordinly (instead of a date he providea a number) then the system displays a messge making the teacher aware of the situation.

15) Register research activities

Description: a teacher wants to register research activities for one of the research projects he is involved in.

Preconditions: the teacher must be authenticated.

Postconditions: if all data is valid the system submits the request to the director, otherwise no request is submited.

15.1) Main Flow

1. The teacher selects the option to request a new research activity.
2. The teacher selects the research project and phase for which he wants to request a new research activity
3. The system displays a form for the teacher to fill
4. The teacher fills the form and submits
5. The system confirms that the operation succeeded

15.2) Alternative flows

14.2.1) Unfilled fields

At 4, the teacher does not fill at least one of the fields then the system displays an allert telling the teacher about the situation.

14.2.2) Invalid data format

At 4, the teacher does not complete at least one of the fields accordinly (instead of a date he providea a number) then the system displays a messge making the teacher aware of the situation.

15) View resource information

Description: a user wantes to see information about all resources (human resourcers, logstical resources etc.).

Preconditions: none.

Postconditions: the system displays a list of resources.

15.1) Main flow

1. The user selects the option to view all resources
2. The system displays all registered resources

15.2) Alternative flows

15.5.1) No registered resources

At 2, there are no registered resources then the system displays a message making the user aware of the situation.

16) View research projects

Description: a user wants to see a list of all research projects that the department has.

Preconditions: none.

Postconditions: the system displays a list of research projects.

16.1) Main flow

1. The user selects the option to view a list of research projects
2. The system displays a list with all active research projects

16.2) Alternative flows

At 2, there are no active research projects then the system displays a message making the user aware of the situation.

17) View the calendar of activities

Description: a user wants to see a calendar of activities that the department has.

Preconditions: none.

Postconditions: the system displays a callendar with the activities it has.

17.1) Main flow

1. The user selects the option to view the calendar of activities the department has.
2. The system displays the calendar of activities.

17.2) Alternative flows

At 2, there are no activities then the system displays a message making the user aware of the situation.

Iteration plan

1) Key milestones

Milestone	Date
Analysis of the problem	6th of March
Application Design	3rd of April
Human, logistical and financial resources registration	10th of April
Defining study programs and imports (schedule, “Stat de funtii”)	17th of April
Defines research project, request for research projects and research activities from the department director	24th of April
Defines administrative events and student research groups, request for administrative events from department director	2nd of May
Approve activities and obtain reports about resource allocation	8th of May
Obtain calendar for upcoming activities and view list of resources (human and logistical)	15th of May

2) High-level objectives

- Present the whole conceptual model and use case realization by 10th of Match
- Present the application architecture sketch by the 1st of April
- Present a working demo by 15th of May

3) Work item assignments

Each team member takes an issue (task) from the feature list and tries to finish it. No records for work hours are made. All tasks are prioritized. One team member may drop the issue if it becomes to difficult or there are dependencies on other components. Once a team member finishes an issue he is responsible for fixing bugs that may appear later, however anyone can take the task into care and fix the bug instead of the team member that finished it.

4) Evaluation criteria

- The conceptual model is simple, sound and easily understood by the team and user.
- The application architecture sketch is sound and covers all possible scenarios that the application may run into (invalid input from the user, errors coming from data layer etc.)
- The working demo satisfies both the team members and the user. Feedback is requested after the demo presentation in order to improve the overall quality of the software.

Project plan

1) Introduction

The project plan illustrates the team structure and shows how the team plans to get the job done in time. Delays are expected since estimations are never 100% precise. The last section "lessons learned" illustrates some small advices for future projects.

2) Team structure

Project manager: Andrei Fangli

User Interface team leader: Gheorghe Ivanciuc Ștefan

User Interface team: Bogdana Mihaela Haiura, Eliza Raluca Cristea, Ioana Elena Cerceș, Roberta Gabriela Haiura

Business Logic team leader: Alexandra Mihălțan

Business Logic team: Codrin Rareș Lache, Eduard Cristian Boloș, Paula Anca Dan

Data Access team leader: Marius Andrei Păsălău

Data Access team: Alexandru Nicolae Iovan, Andrea Rado

3) Responsibilities

- The project manager is responsible with problem analysis, application design and aiding all team members when needed. The project manager should make the modeling part of the development process to avoid developing a system based on a bad design (sometimes the quality of the design can only be evaluated at development time by the developers).
- The User Interface team is responsible with designing a rich user interface and prototyping. The prototype is used as a fast development process where no binding with the actual application implementation is done to allow the user to visualize how his application will look like and provide feedback. Having an early feedback from the user can confirm that the problem was understood correctly and if not it can be easily adjusted since no real development effort was done.
- The Business Logic team is responsible with creating the static structure of the design (implement all entities and validation logic) and then provide business workflow components (controllers, custom validators etc.). Also the mapping between objects must be implemented as it is in the conceptual model with respect to all associations (directed and undirected) and consistency.
- The Data Access team is responsible with transforming (marshaling) objects and object associations into a persistent, physical state which can be kept on an external memory. The saved data must also be able to be loaded. At this level a relational database is usually used because such tools allow to keep relations between tables and execute complex queries without having to actually implement the logic behind each query.

4) Deployment

The application is planned to be deployed "all at once", in other words there is no versioning for the application, future releases may include only bug fixes, however there is always room for extension. The reason for this approach is that the user asked for all features and agreed to not change the requirements of the software during it's development.

5) Lessons learned

"Never put off till tomorrow what you can do today", in our case "Never put off till next month what you can do today". We had trouble managing our time and later realized that there isn't much left and we had to "kick it", and fast. One thing is sure about the experience of this project, never put off something. The sooner the team meets and starts the development the sooner the project manager gets feedback on his model and can adjust it without having to pay an expensive price in time. The later it gets the more value time has. Sacrifice a few weekends if needed, but get your head straight as soon as possible and jump on that keyboard right away if you don't want to end up giving up more weekends and more nights developing in the last weeks under time pressure and lack of faith from the user you have committed to.

Architecture notebook

1) Architectural goals and philosophy

The architectural philosophy is centered around the “Single responsibility principle” where any class is designed to handle a single responsibility in order to decouple tight relations between objects and allow maximum code reuse. The degree of complexity is decreased and testing is made even easier and in some cases not required since some components are very easy to implement and do not require testing (simple entities such as Color, Shape, ClassRoom etc.).

The goal of the development is to use the most common approach to all members. If all members followed a databases course where they learned to use ADO.NET then the DataAccess objects are based on ADO.NET since, theoretically, anyone in the team can pick a task or a bug and understand what’s going on and greatly increase the chances of fixing the bug or resolving the task rather than waiting on the few that have experience with other tools such as ORMs.

2) Assumptions and dependencies

The team is assumed to have knowledge with text file manipulation and relational database query (SQL), basic knowledge of object oriented paradigm and basic understanding of Unified Modeling Language (UML). All architectural decisions are made based on this assumption and if there is a way to achieve better performance using a functional programming approach or a Not Only SQL (noSQL) it is ignored because the team is considered to not have knowledge or experience with such tools.

3) Architectural mechanisms

The architecture tries to make use of the MVVM (Model-View-ViewModel) pattern in order to separate the display logic from the business logic. In this pattern and with this architecture, the Model is a class defined in the Entities sub-layer, the View is a component defined in the UserInterface layer and the ViewModel is a Business Workflow component. The ViewModel acts as a controller and decides what happens when receiving events and also acts as a value converter, it can convert raw input (such as strings) into Entities (such as a ClassRoom).

The communication between the UserInterface and Business Workflow components is made using the Try... pattern. In this pattern all methods prefixed with “Try” will rather try to do the requested job and return a Boolean value telling w hether the operation failed or succeeded instead of throwing an exceptions. The error message is retrieved using an output string parameter which is set only if the method returns false, this parameter holds a user friendly message telling what went wrong. This pattern is ideal for communication between UI and Workflow components since the decision is made in the Workflow component. This pattern is useful because the UI does not need to know what to with Exception objects since all it receives is a Boolean value and an output string parameter containing the error message (if any).

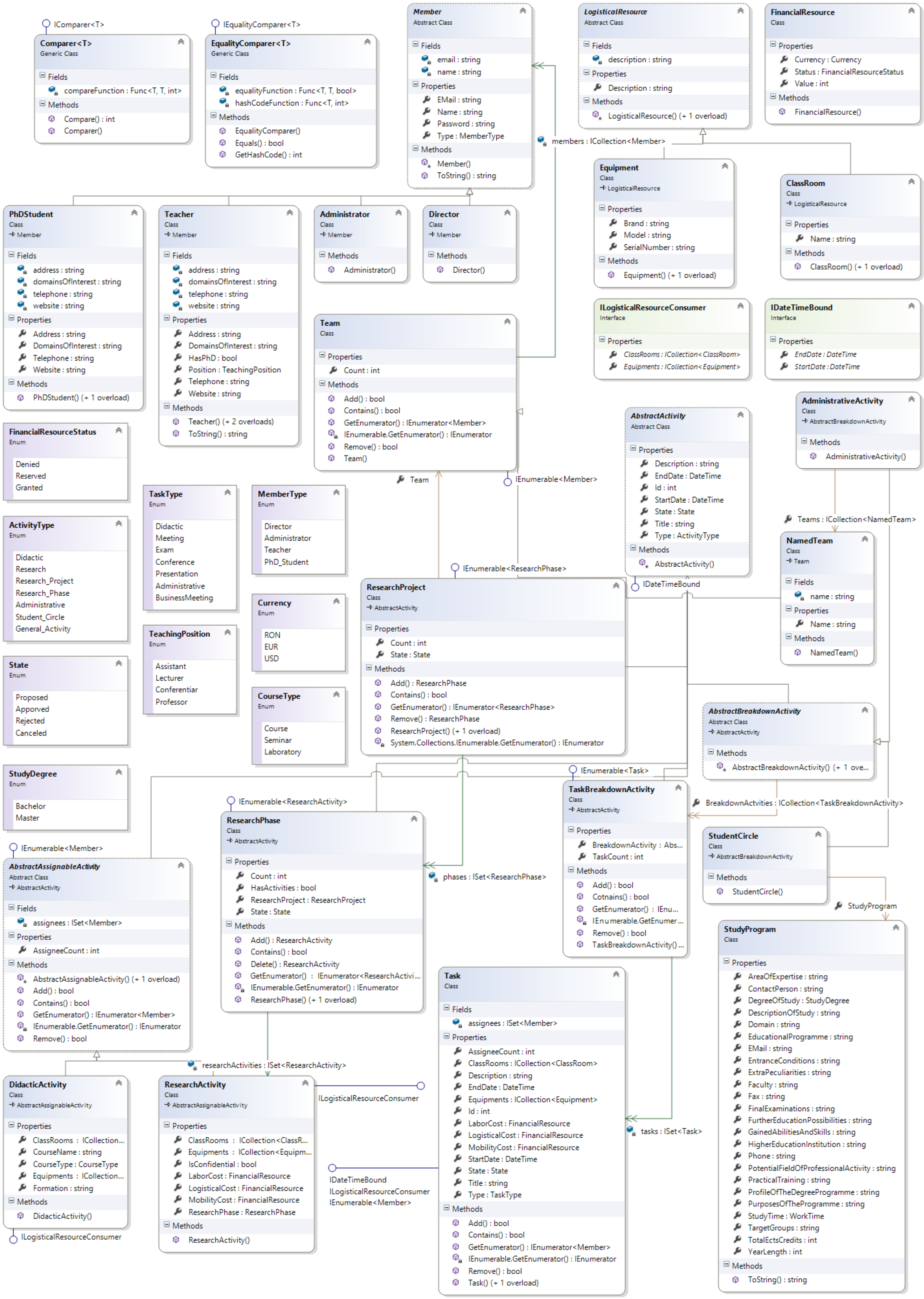
4) Layers in architecutre

The architecture is divided into layers: UserInterface, BusinessLogic and DataAccess. Each layer is described by its name, the UserInterface layer is responsible only with dealing with the user and nothing more. Components in this layer only display data to the user and retrieve user input (mouse clicks, key press etc.) and sends them to components found in the BusinessLogic layer. No decision making components can be found in the UserInterface layer.

The BusinessLogic layer is divided intro three sub-layers: Entities, DAOInterface and Workflow. The Entities sub-layer contains all definitions of entities, typically the implementation of the conceptual model and its associations alone. This brings the benefit of creating a project that contains only the entities which can be referenced from anywhere from the application, any design patterns can be applied in this case. The 2nd sub-layer is the DAOInterface, this is nothing more but a collection of interfaces that Workflow components use. Finally, the 3rd sub-layer, Workflow, contains components that are responsible with decision making. Once a signal is retrieved from the user, a Workflow component takes it into care and decides what the software should do for such an event. The result can be a method call to one of the DAOInterface objects, a creation of an entity etc. The Workflow components are also responsible for getting and sending data to a persistent state. This is done through the DAOInterfaces (DataAccessObjectInterfaces), for any DataAccess implementation to be fully compatible with the application it must correctly implement all DAOInterfaces.

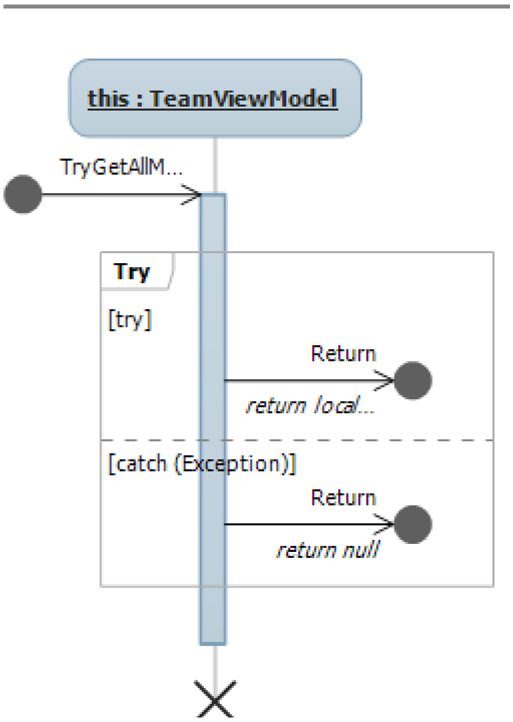
Last but not least the DataAccess layer is responsible with data access, all components at this level implement at least one of the DAOInterfaces defined in the BusinessLogic layer. This layer does not contain sub-layers by default, however it is possible to define different persistence marshaling components such as saving data to text files, XML files, relational database etc.

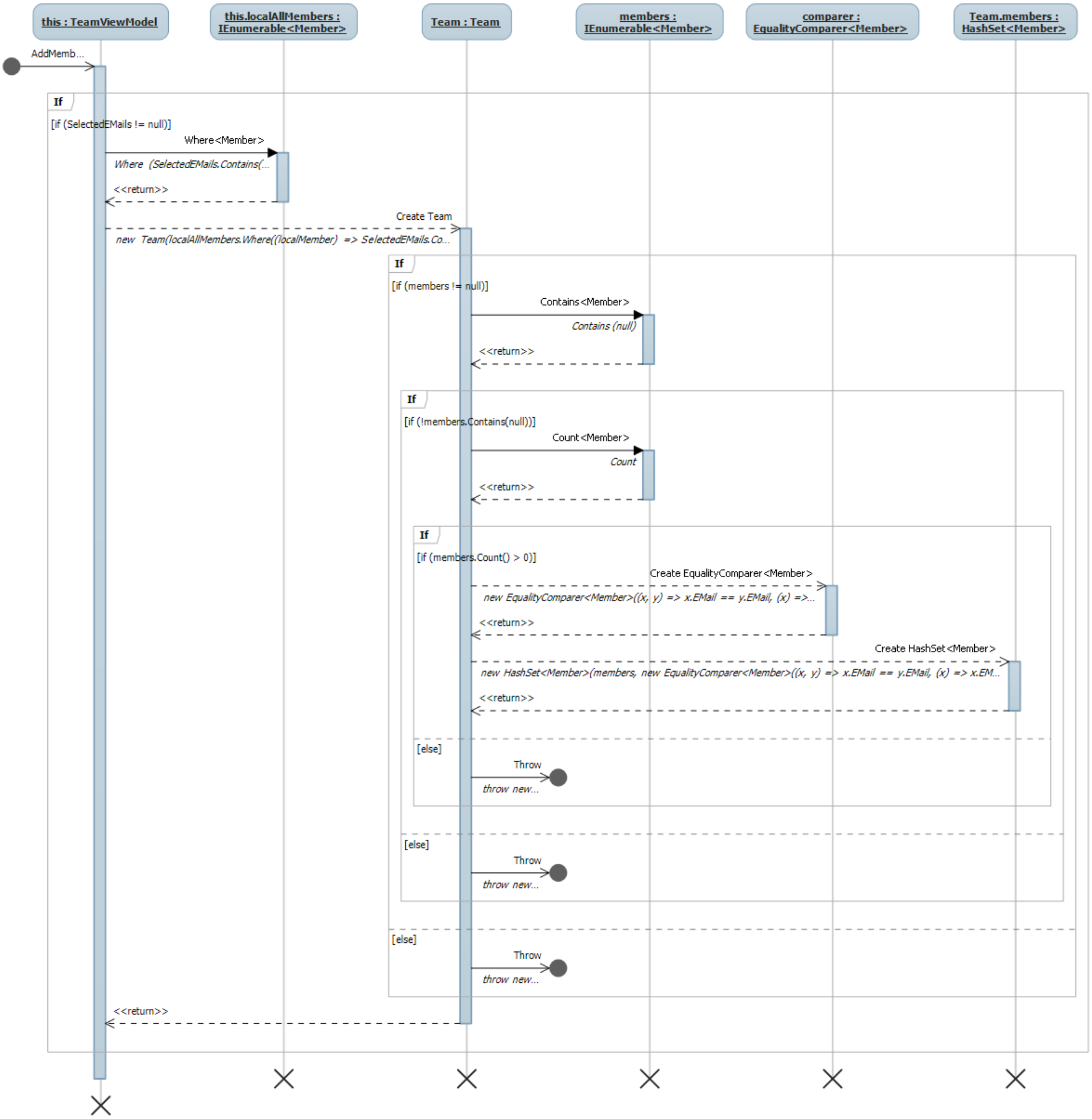
5) Conceptual Model

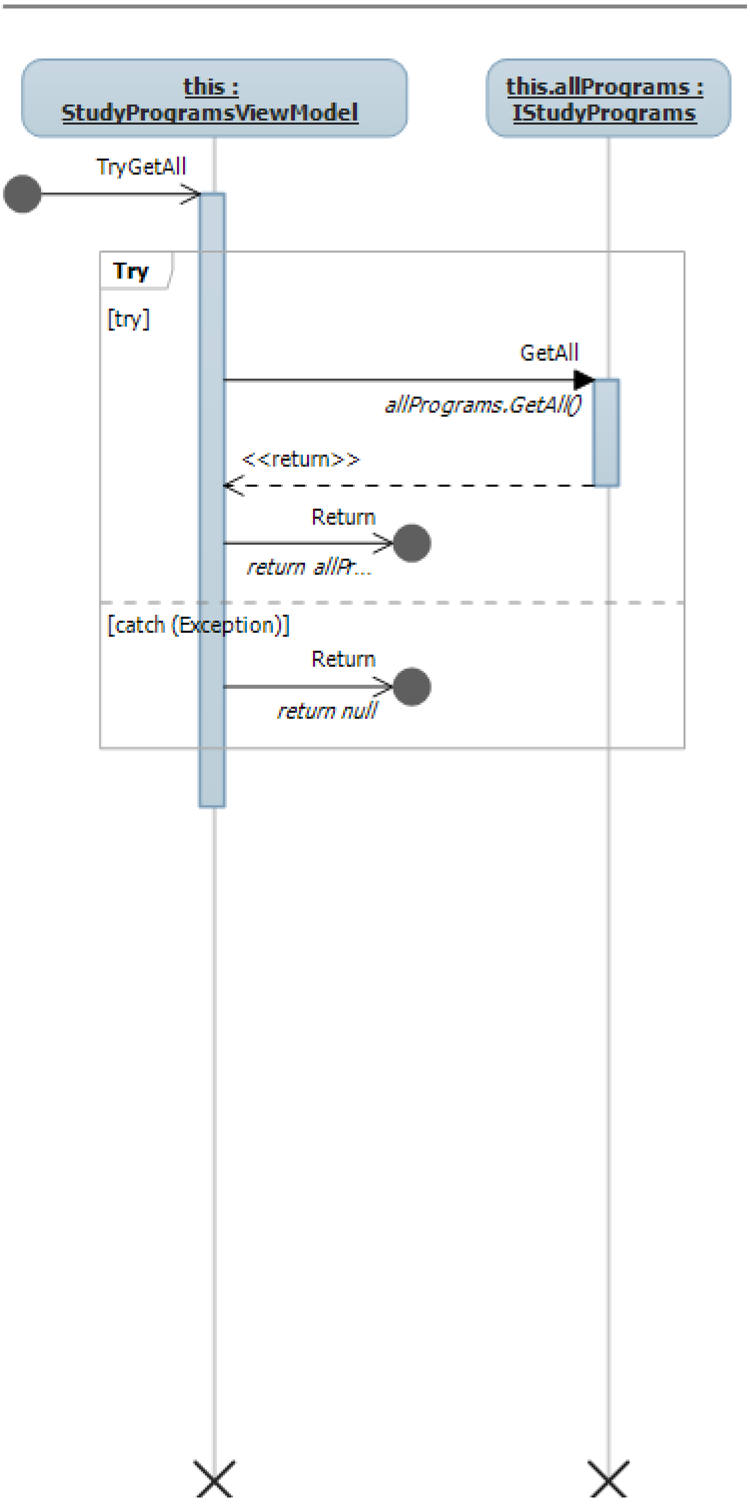


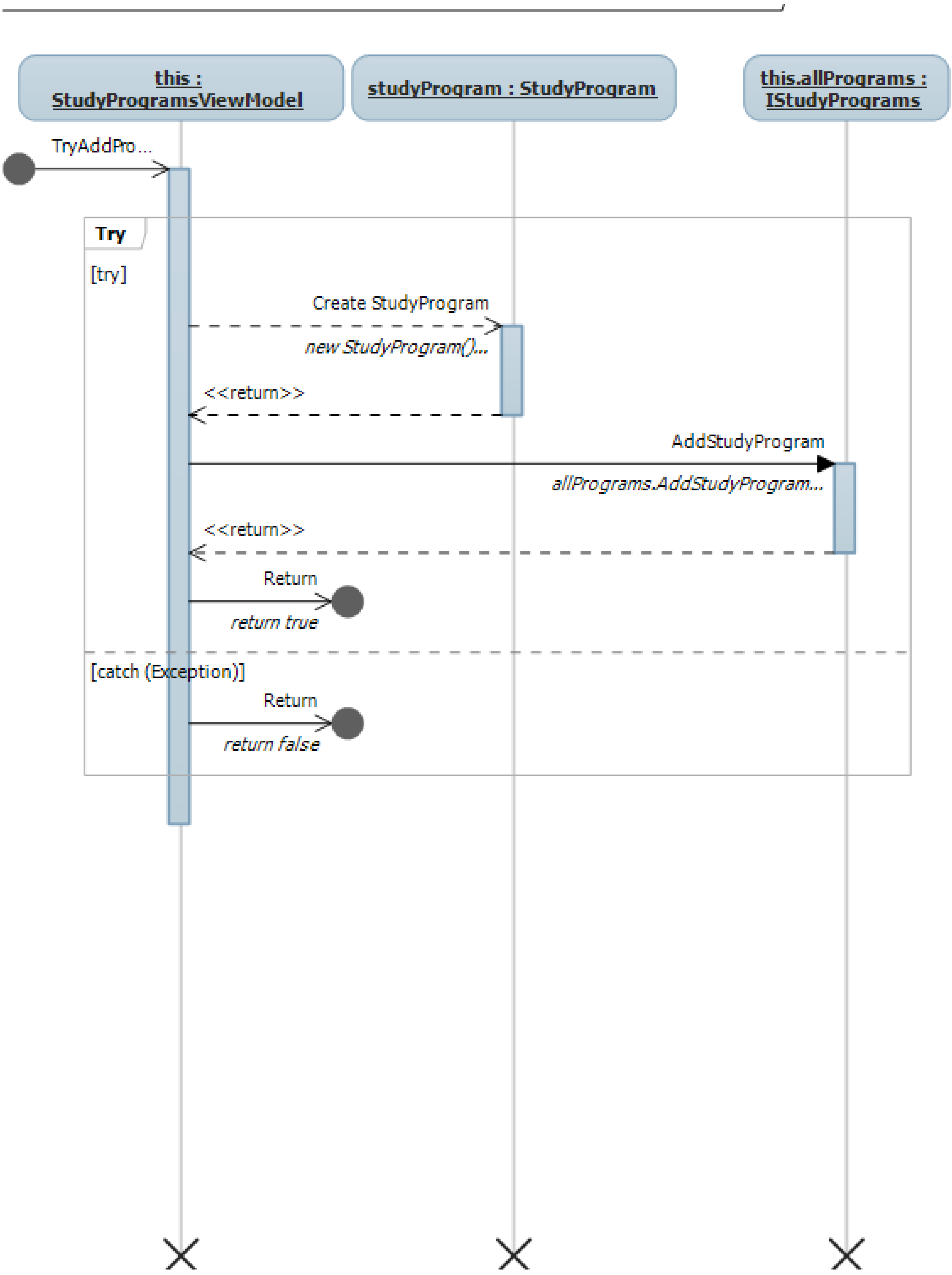
Design

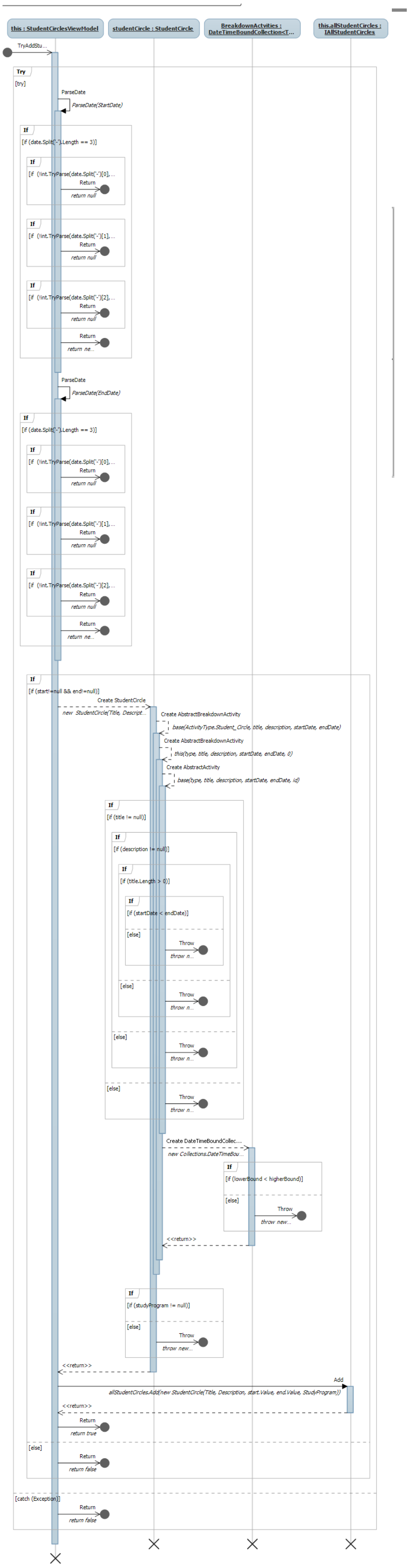
1) Use case realization

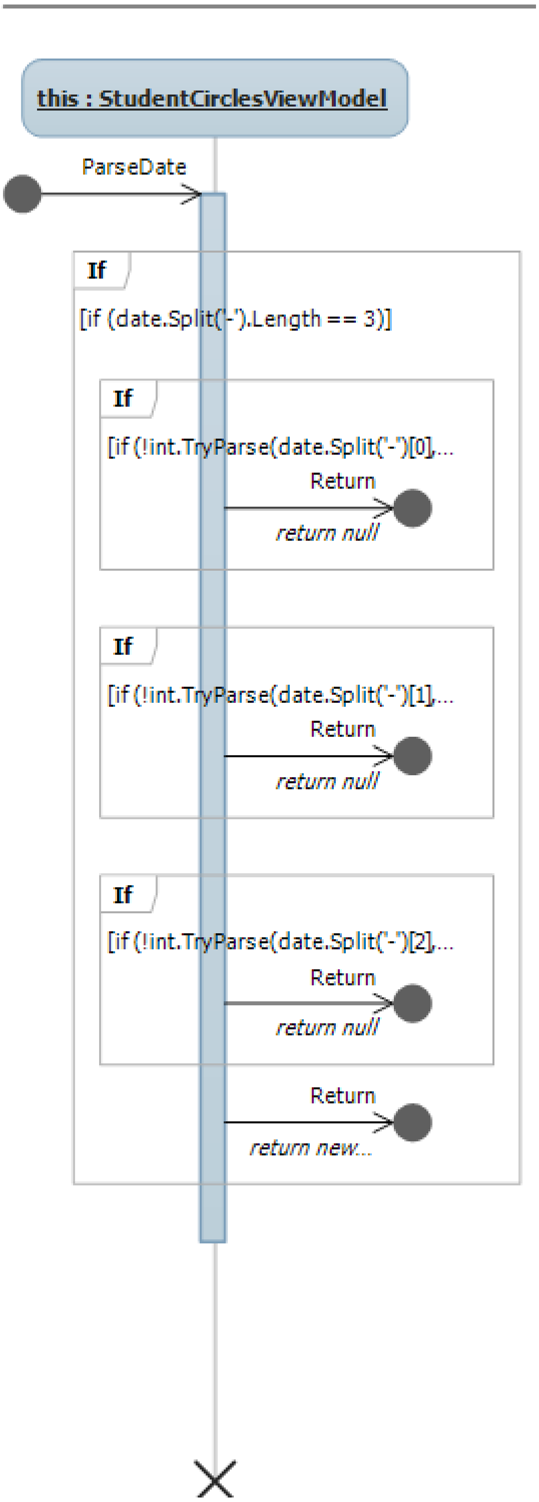


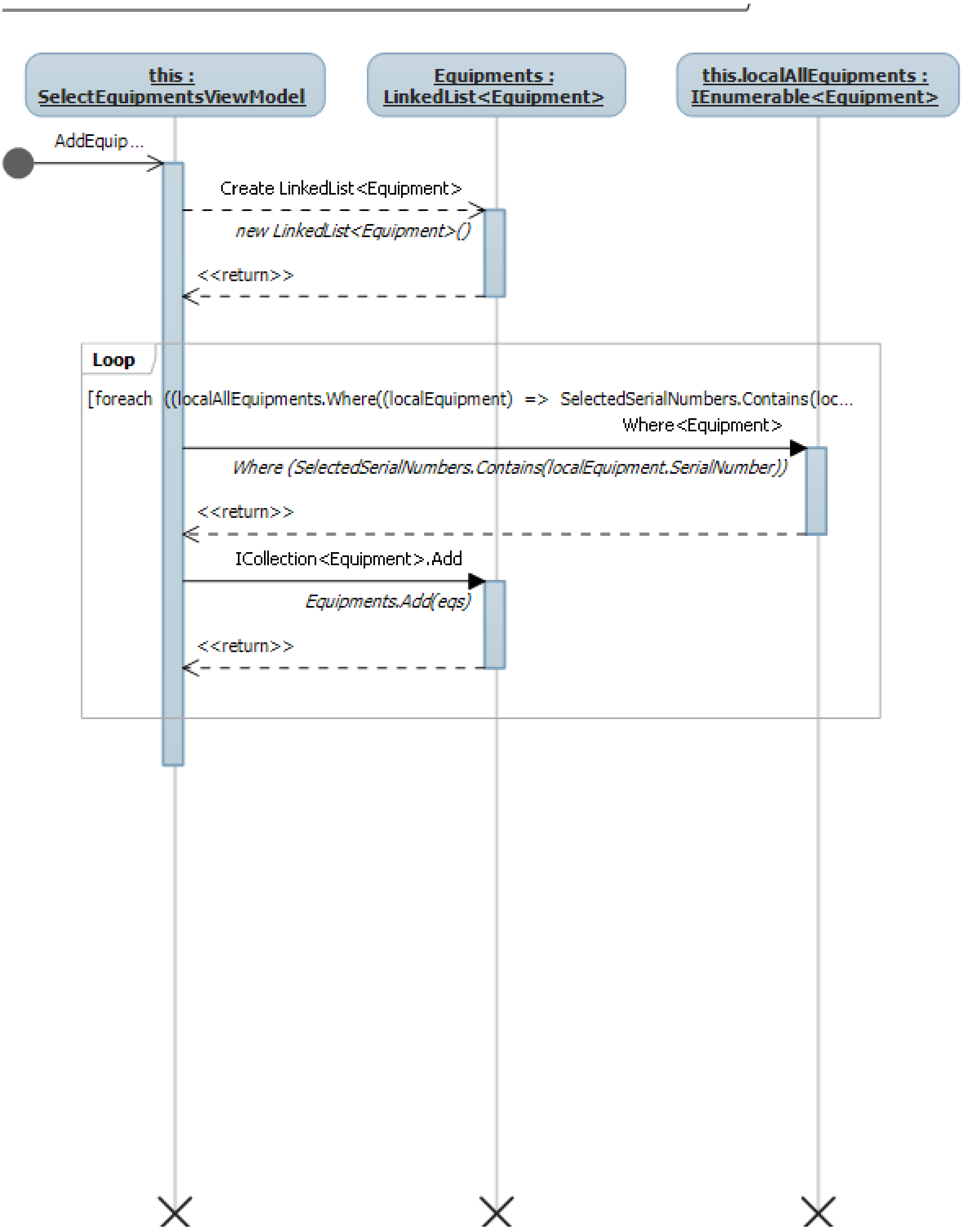

this : TeamViewModel
this.localAllMembers : IEnumerable<Member>
Team : Team
members : IEnumerable<Member>
comparer : EqualityComparer<Member>
Team.members : HashSet<Member>
AddMemb...
If
[if (SelectedEMails != null)]
Where<Member>
Where (SelectedEMails.Contains(...
<<return>>
Create Team
new Team(localAllMembers.Where((localMember) => SelectedEMails.Co...
If
[if (members != null)]
Contains<Member>
Contains (null)
<<return>>
If
[if (!members.Contains(null))]
Count<Member>
Count
<<return>>
If
[if (members.Count() > 0)]
Create EqualityComparer<Member>
new EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) =>...
<<return>>
Create HashSet<Member>
new HashSet<Member>(members, new EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) => x.EM...
<<return>>
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
<<return>>

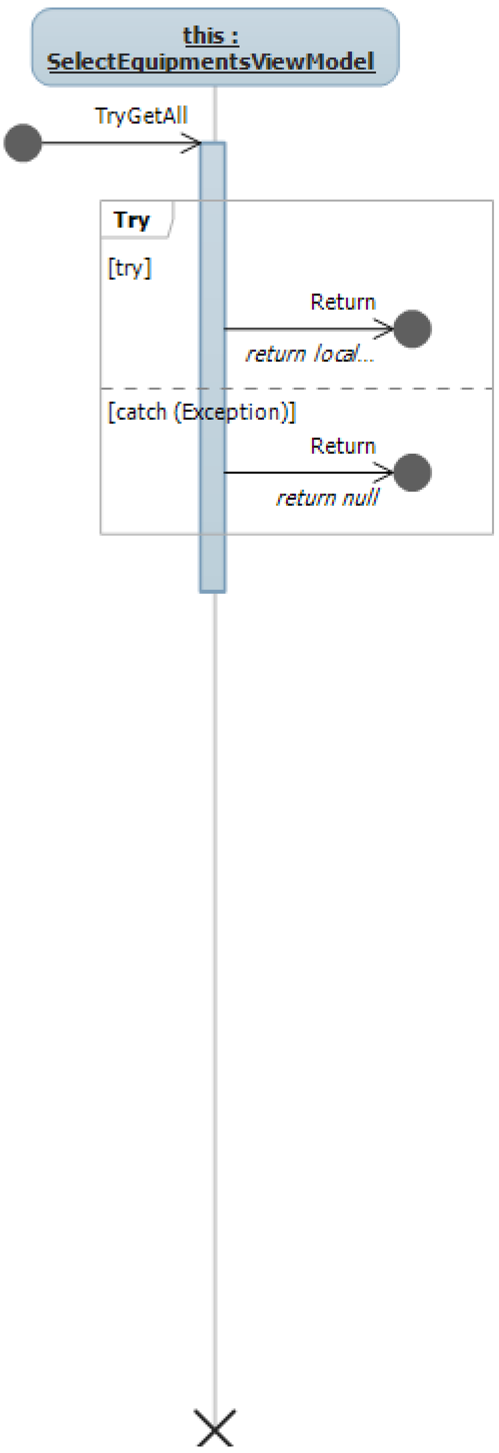

this : StudyProgramsViewModel
this.allPrograms : IStudyPrograms
TryGetAll
Try
[try]
GetAll
allPrograms.GetAll()
<<return>>
Return
return allPr...
[catch (Exception)]
Return
return null

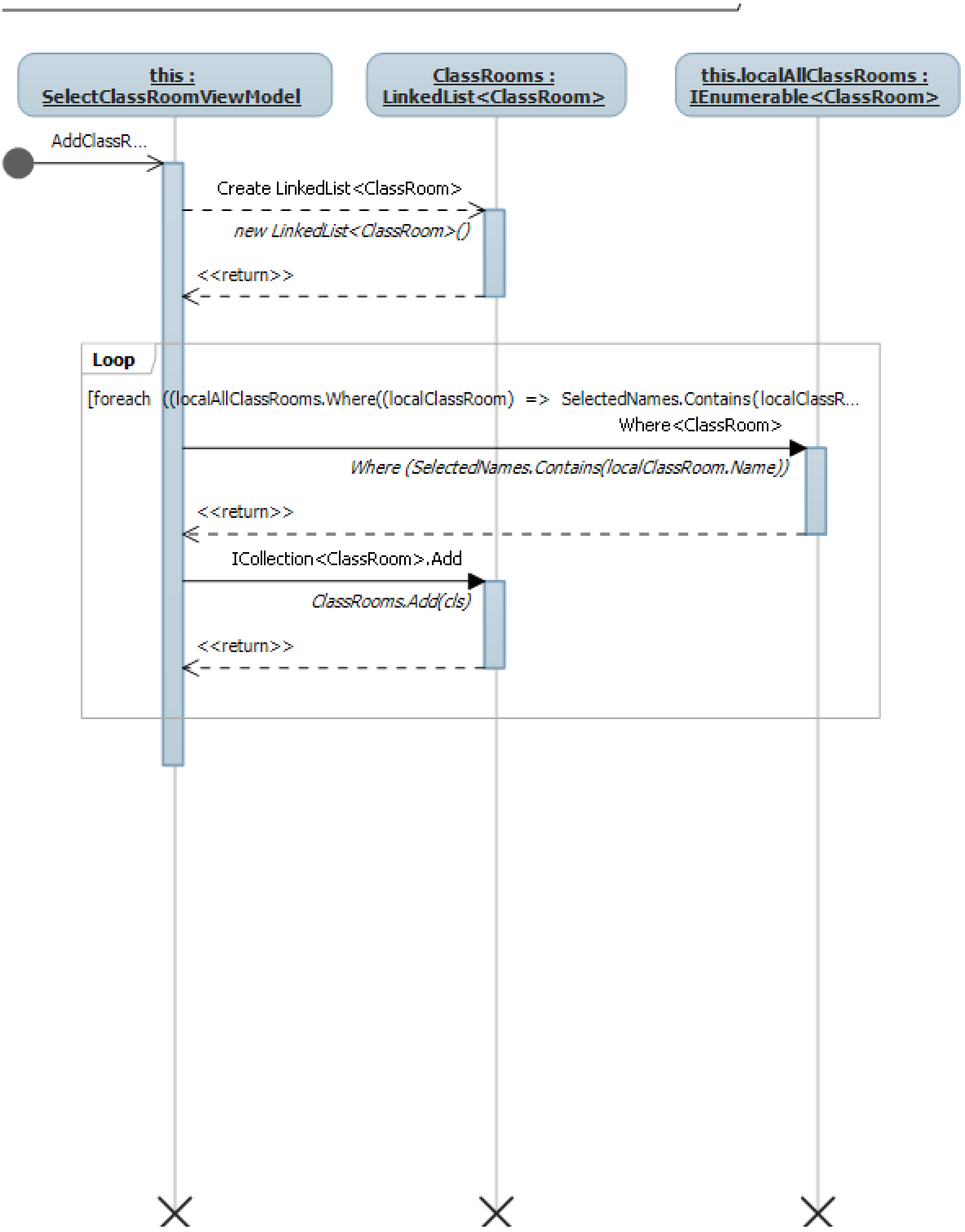

this : StudyProgramsViewModel
studyProgram : StudyProgram
this.allPrograms : IStudyPrograms
TryAddPro...
Try
[try]
Create StudyProgram
new StudyProgram()...
<<return>>
AddStudyProgram
allPrograms.AddStudyProgram...
<<return>>
Return
return true
[catch (Exception)]
Return
return false

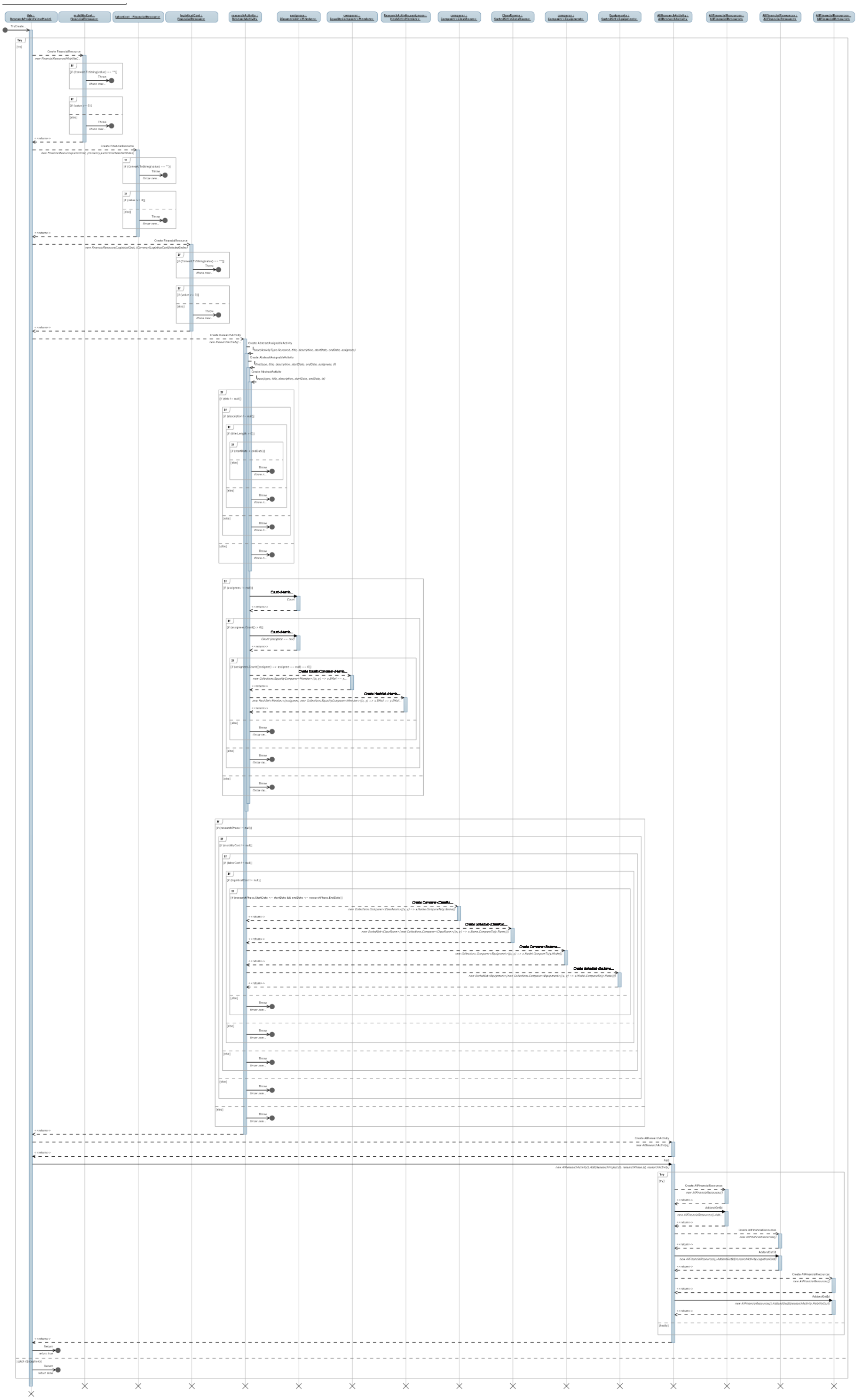

this : StudentCirclesViewModel
studentCircle : StudentCircle
BreakdownActvities : DateTimeBoundCollection<T...
this.allStudentCircles : IAllStudentCircles
TryAddStu...
Try
[try]
ParseDate
ParseDate(StartDate)
If
[if (date.Split('-').Length == 3)]
If
[if (!int.TryParse(date.Split('-')[0],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[1],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[2],...
Return
return null
Return
return ne...
ParseDate
ParseDate(EndDate)
If
[if (date.Split('-').Length == 3)]
If
[if (!int.TryParse(date.Split('-')[0],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[1],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[2],...
Return
return null
Return
return ne...
If
[if (start!=null && end!=null)]
Create StudentCircle
new StudentCircle(Title, Descript...
Create AbstractBreakdownActivity
base(ActivityType.Student_Circle, title, description, startDate, endDate)
Create AbstractBreakdownActivity
this(type, title, description, startDate, endDate, 0)
Create AbstractActivity
base(type, title, description, startDate, endDate, id)
If
[if (title != null)]
If
[if (description != null)]
If
[if (title.Length > 0)]
If
[if (startDate < endDate)]
[else]
Throw
throw n...
[else]
Throw
throw n...
[else]
Throw
throw n...
[else]
Throw
throw n...
Create DateTimeBoundCollec...
new Collections.DateTimeBou...
If
[if (lowerBound < higherBound)]
[else]
Throw
throw new...
<<return>>
If
[if (studyProgram != null)]
[else]
Throw
throw new...
<<return>>
Add
allStudentCircles.Add(new StudentCircle(Title, Description, start.Value, end.Value, StudyProgram))
<<return>>
Return
return true
[else]
Return
return false
[catch (Exception)]
Return
return false

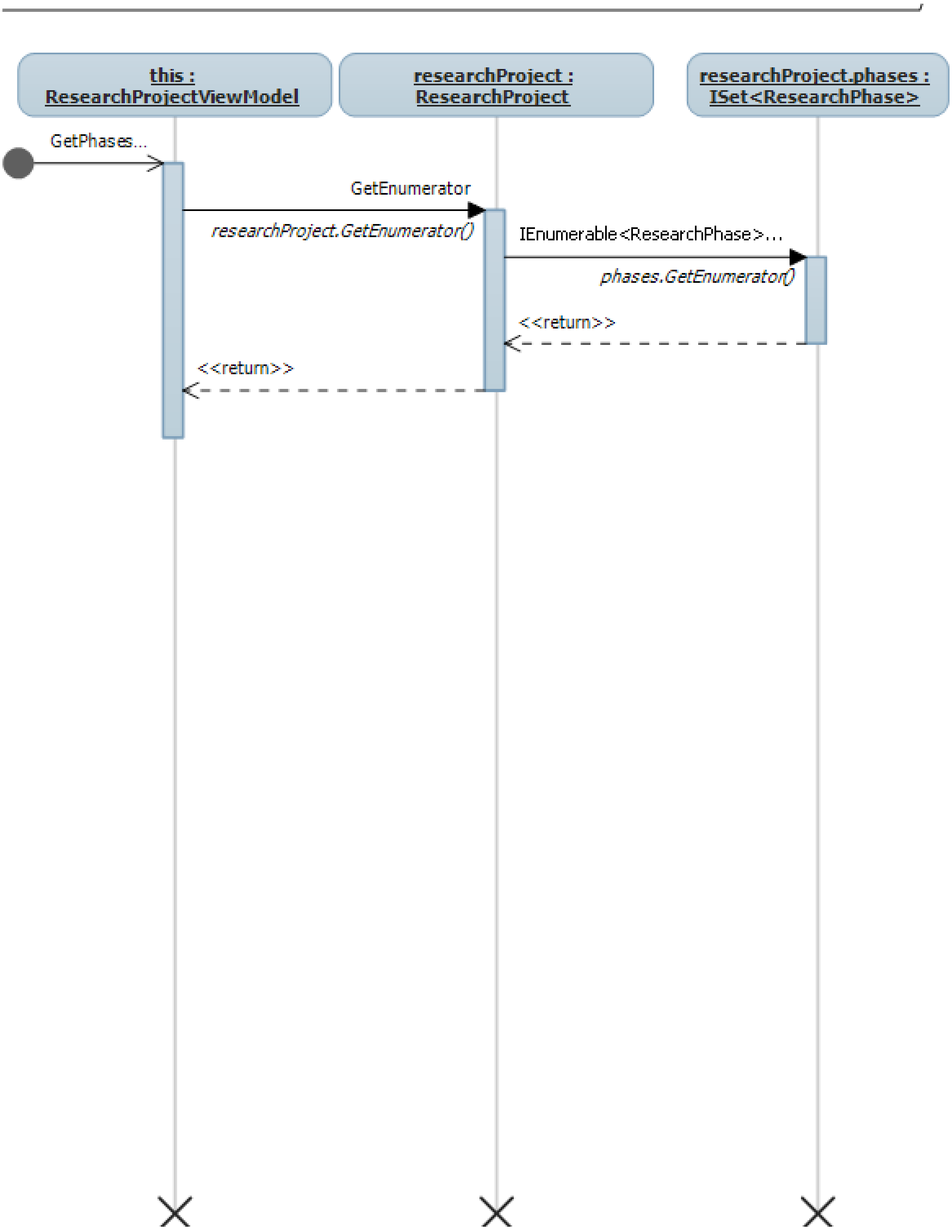

this : StudentCirclesViewModel
ParseDate
If
[if (date.Split('-').Length == 3)]
If
[if (!int.TryParse(date.Split('-')[0],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[1],...
Return
return null
If
[if (!int.TryParse(date.Split('-')[2],...
Return
return null
Return
return new...

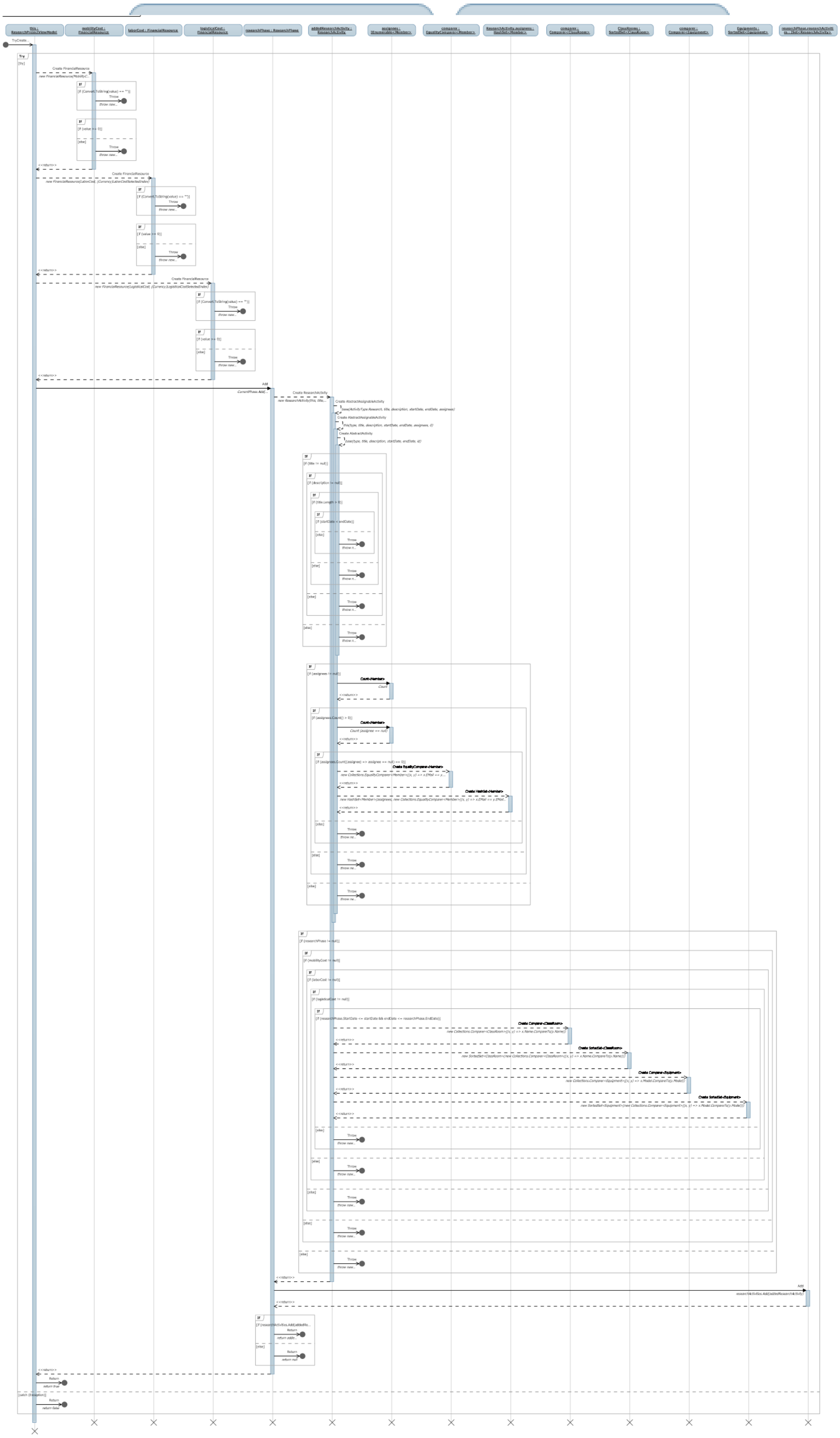

this : SelectEquipmentsViewModel
Equipments : LinkedList<Equipment>
this.localAllEquipments : IEnumerable<Equipment>
AddEquip...
Create LinkedList<Equipment>
new LinkedList<Equipment>()
<<return>>
Loop
[foreach ((localAllEquipments.Where((localEquipment) => SelectedSerialNumbers.Contains(loc...
Where<Equipment>
Where (SelectedSerialNumbers.Contains(localEquipment.SerialNumber))
<<return>>
ICollection<Equipment>.Add
Equipments.Add(eqs)
<<return>>

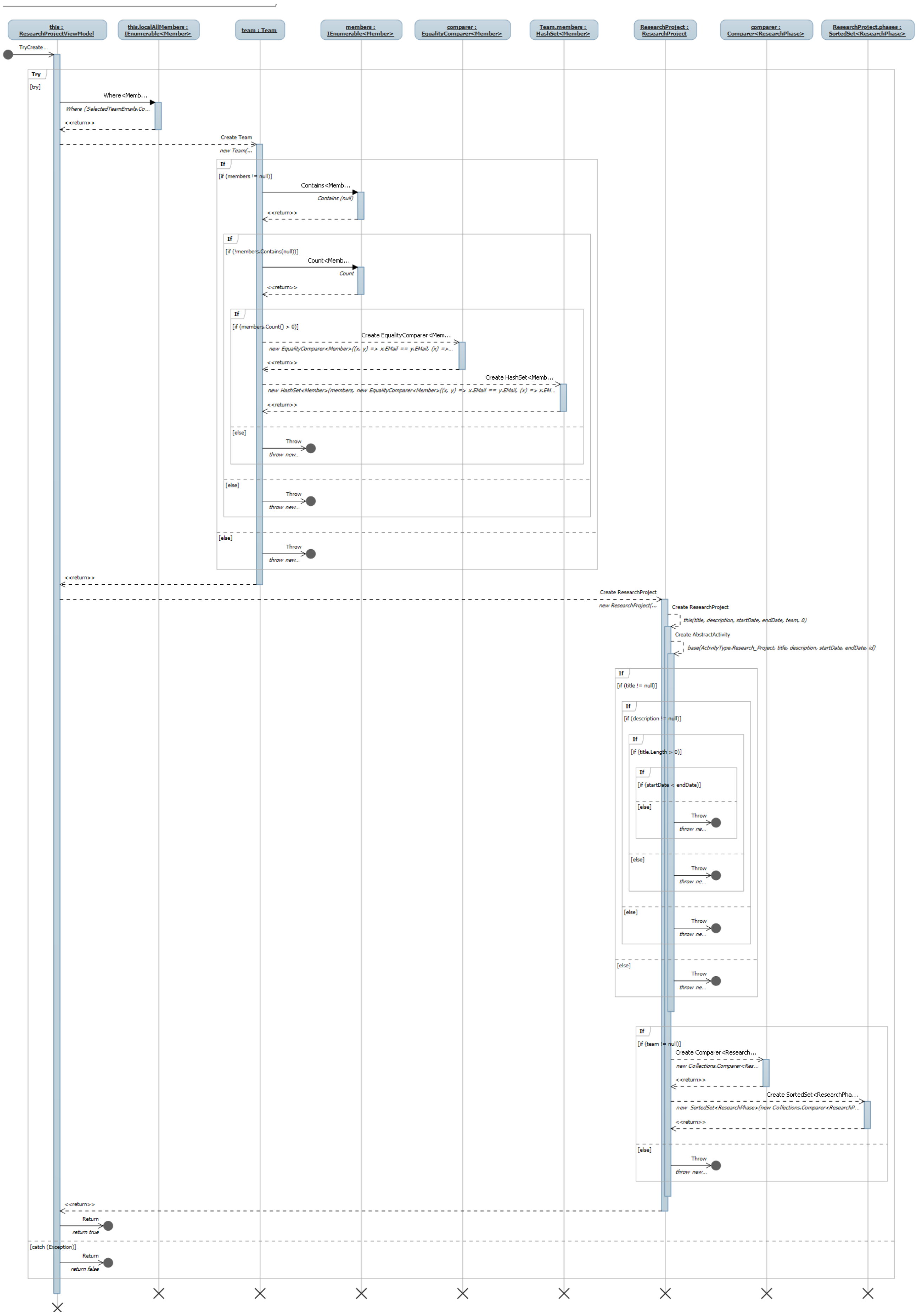

this : SelectEquipmentsViewModel
TryGetAll
Try
[try]
Return
return local...
[catch (Exception)]
Return
return null

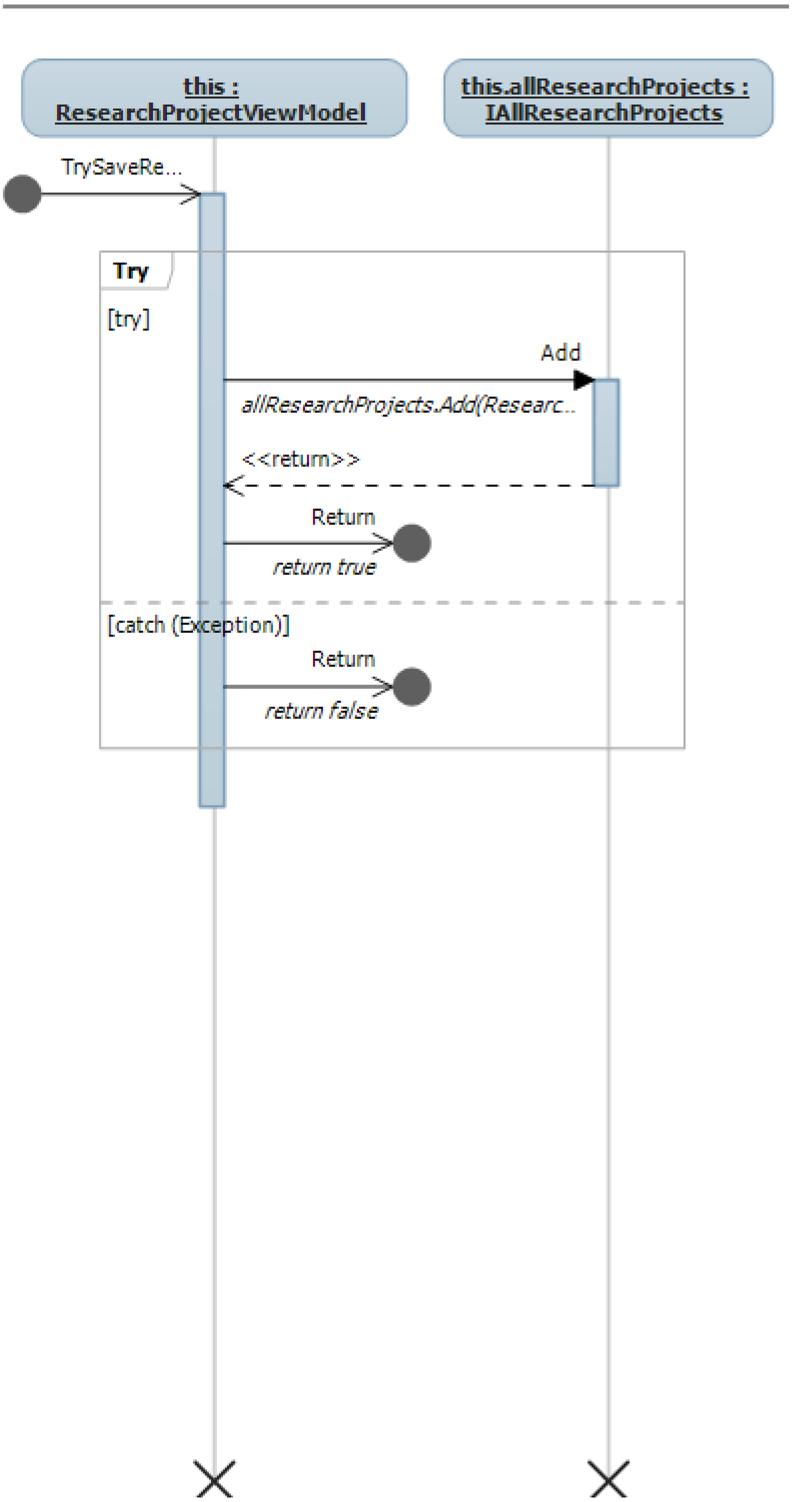

this : SelectClassRoomViewModel
ClassRooms : LinkedList<ClassRoom>
this.localAllClassRooms : IEnumerable<ClassRoom>
AddClassR...
Create LinkedList<ClassRoom>
new LinkedList<ClassRoom>()
<<return>>
Loop
[foreach ((localAllClassRooms.Where((localClassRoom) => SelectedNames.Contains(localClassR...
Where<ClassRoom>
Where (SelectedNames.Contains(localClassRoom.Name))
<<return>>
ICollection<ClassRoom>.Add
ClassRooms.Add(cls)
<<return>>

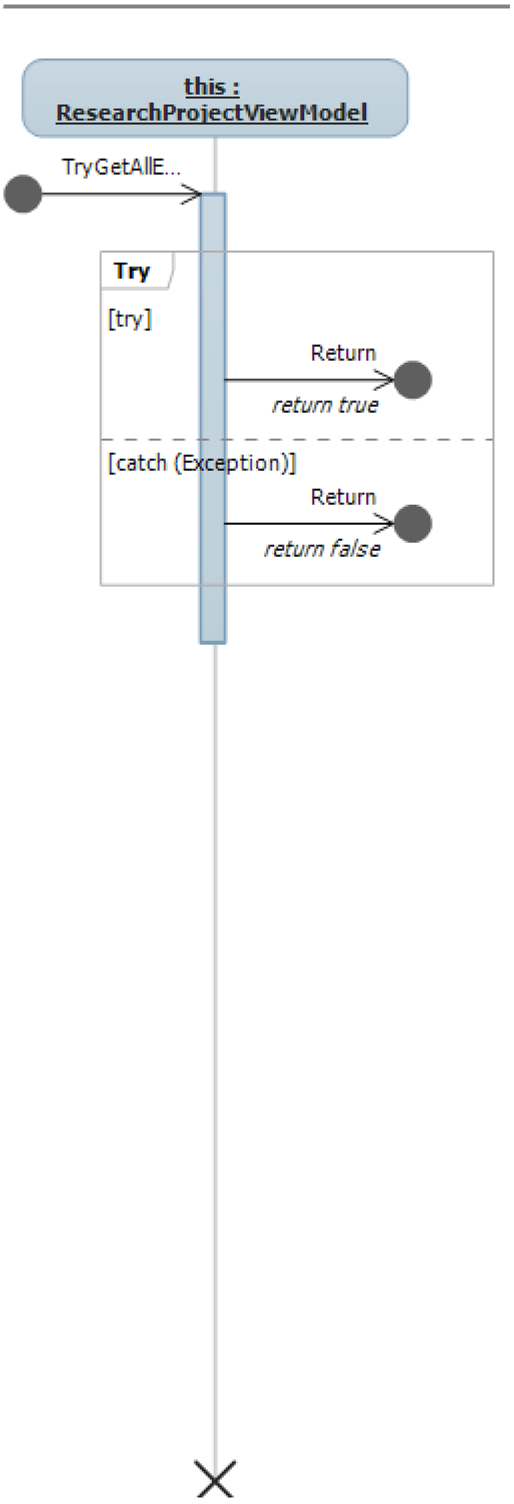

this : ResearchProjectViewModel
mobilityCost : FinancialResource
laborCost : FinancialResource
logisticalCost : FinancialResource
researchActivity : ResearchActivity
assignees : IEnumerable<Member>
comparer : EqualityComparer<Member>
ResearchActivity.assignees : HashSet<Member>
comparer : Comparer<ClassRoom>
ClassRooms : SortedSet<ClassRoom>
comparer : Comparer<Equipment>
Equipments : SortedSet<Equipment>
AllResearchActivity : AllResearchActivity
AllFinancialResources : AllFinancialResources
AllFinancialResources : AllFinancialResources
AllFinancialResources : AllFinancialResources
TryCreate...
Try
[try]
Create FinancialResource
new FinancialResource(MobilityC...
If
[if (Convert.ToString(value) == "")]
Throw
throw new...
[if (value >= 0)]
[else]
<<return>>
Create ResearchActivity
Create AllResearchActivity
new AllResearchActivity()
Add
new AllResearchActivity().Add(ResearchProject.Id, researchPhase.Id, researchActivity)
Create AllFinancialResources
new AllFinancialResources()
AddandGetId
new AllFinancialResources().Add...
new AllFinancialResources().AddandGetId(researchActivity.LogisticalCost)
new AllFinancialResources().AddandGetId(researchActivity.MobilityCost)
[finally]
Return
return true
[catch (Exception)]
return false

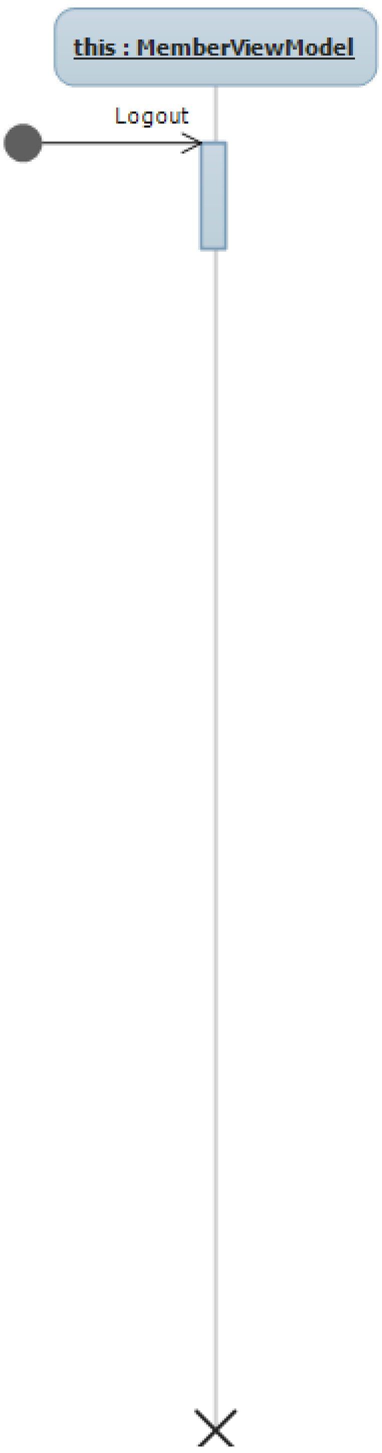

this : ResearchProjectViewModel
researchProject : ResearchProject
researchProject.phases : ISet<ResearchPhase>
GetPhases...
GetEnumerator
researchProject.GetEnumerator()
IEnumerable<ResearchPhase>...
phases.GetEnumerator()
<<return>>
<<return>>

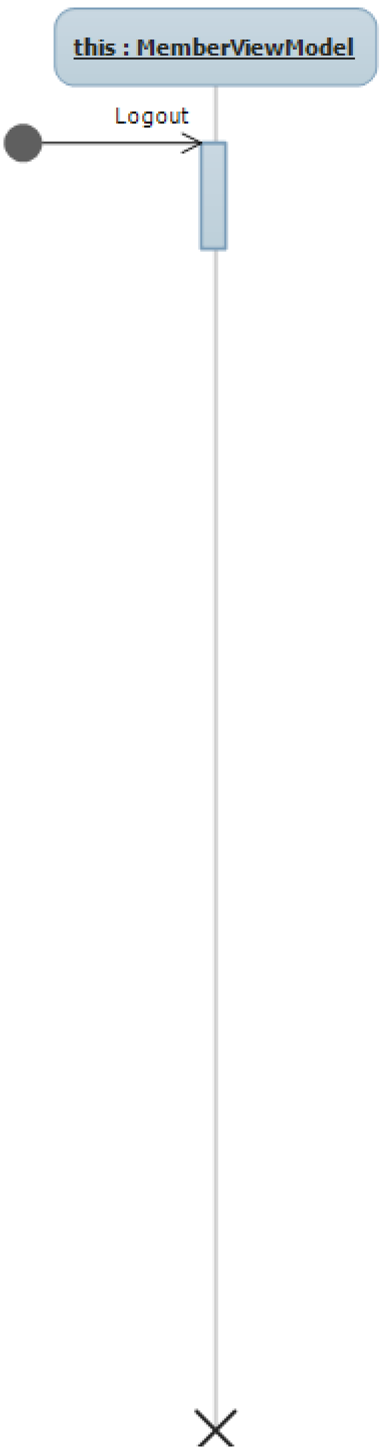

this : ResearchProjectViewModel
mobilityCost : FinancialResource
laborCost : FinancialResource
logisticalCost : FinancialResource
researchPhase : ResearchPhase
addedResearchActivity : ResearchActivity
assignees : IEnumerable<Member>
comparer : EqualityComparer<Member>
ResearchActivity.assignees : HashSet<Member>
comparer : Comparer<ClassRoom>
ClassRooms : SortedSet<ClassRoom>
comparer : Comparer<Equipment>
Equipments : SortedSet<Equipment>
researchPhase.researchActivities : ISet<ResearchActivity>
TryCreate...
Try
[try]
Create FinancialResource
new FinancialResource(MobilityC...
If
[If (Convert.ToString(value) == "")]
Throw
throw new...
[If (value >= 0)]
[else]
<<return>>
Add
CurrentPhase.Add(...
Create ResearchActivity
new ResearchActivity(this, title,...
Create AbstractAssignableActivity
base(ActivityType.Research, title, description, startDate, endDate, assignees)
this(type, title, description, startDate, endDate, assignees, 0)
Create AbstractActivity
base(type, title, description, startDate, endDate, id)
[If (title != null)]
[If (description != null)]
[If (title.Length > 0)]
[If (startDate < endDate)]
[If (assignees != null)]
Count<Member>
Count
[If (assignees.Count() > 0)]
Count (assignee == null)
[If (assignees.Count((assignee) => assignee == null) == 0)]
Create EqualityComparer<Member>
new Collections.EqualityComparer<Member>((x, y) => x.EMail == y....
Create HashSet<Member>
new HashSet<Member>(assignees, new Collections.EqualityComparer<Member>((x, y) => x.EMail == y.EMail...
[If (researchPhase != null)]
[If (mobilityCost != null)]
[If (laborCost != null)]
[If (logisticalCost != null)]
[If (researchPhase.StartDate <= startDate && endDate <= researchPhase.EndDate)]
Create Comparer<ClassRoom>
new Collections.Comparer<ClassRoom>((x, y) => x.Name.CompareTo(y.Name))
Create SortedSet<ClassRoom>
new SortedSet<ClassRoom>(new Collections.Comparer<ClassRoom>((x, y) => x.Name.CompareTo(y.Name)))
Create Comparer<Equipment>
new Collections.Comparer<Equipment>((x, y) => x.Model.CompareTo(y.Model))
Create SortedSet<Equipment>
new SortedSet<Equipment>(new Collections.Comparer<Equipment>((x, y) => x.Model.CompareTo(y.Model)))
researchActivities.Add(addedResearchActivity)
[If (researchActivities.Add(addedRe...
Return
return adde...
return null
return true
[catch (Exception)]
return false

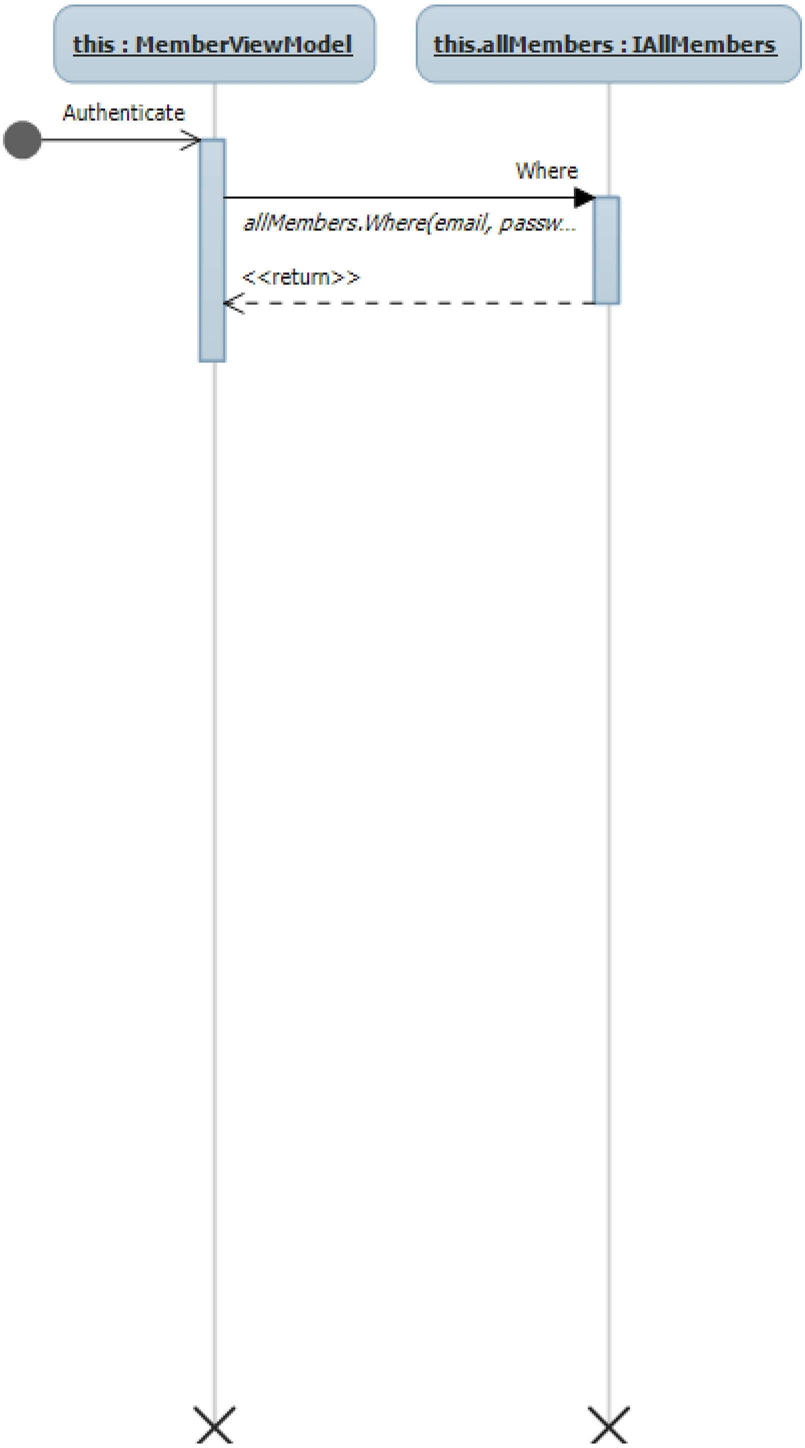

this : ResearchProjectViewModel
this.localAllMembers : IEnumerable<Member>
team : Team
members : IEnumerable<Member>
comparer : EqualityComparer<Member>
Team.members : HashSet<Member>
ResearchProject : ResearchProject
comparer : Comparer<ResearchPhase>
ResearchProject.phases : SortedSet<ResearchPhase>
TryCreate...
Try
[try]
Where<Memb...
Where (SelectedTeamEmails.Co...
<<return>>
Create Team
new Team(...
If
[if (members != null)]
Contains<Memb...
Contains (null)
<<return>>
If
[if (!members.Contains(null))]
Count<Memb...
Count
<<return>>
If
[if (members.Count() > 0)]
Create EqualityComparer<Mem...
new EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) =>...
<<return>>
Create HashSet<Memb...
new HashSet<Member>(members, new EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) => x.EM...
<<return>>
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
<<return>>
Create ResearchProject
new ResearchProject(...
Create ResearchProject
this(title, description, startDate, endDate, team, 0)
Create AbstractActivity
base(ActivityType.Research_Project, title, description, startDate, endDate, id)
If
[if (title != null)]
If
[if (description != null)]
If
[if (title.Length > 0)]
If
[if (startDate < endDate)]
[else]
Throw
throw ne...
[else]
Throw
throw ne...
[else]
Throw
throw ne...
[else]
Throw
throw ne...
If
[if (team != null)]
Create Comparer<Research...
new Collections.Comparer<Res...
<<return>>
Create SortedSet<ResearchPha...
new SortedSet<ResearchPhase>(new Collections.Comparer<ResearchP...
<<return>>
[else]
Throw
throw new...
<<return>>
Return
return true
[catch (Exception)]
Return
return false

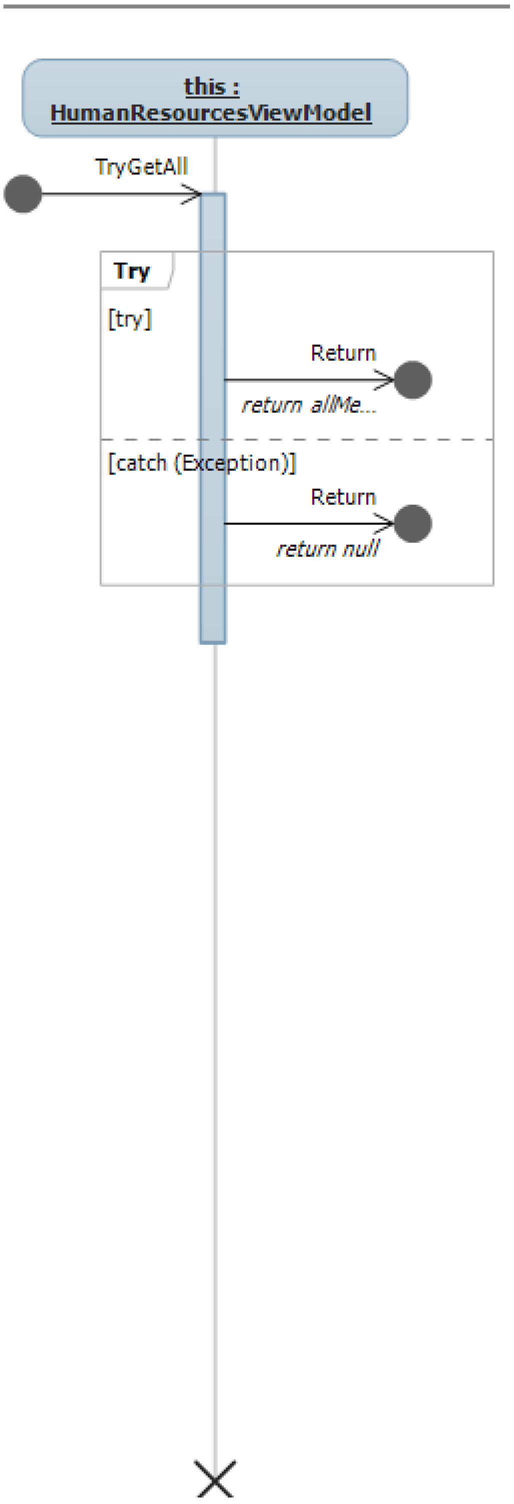

this : ResearchProjectViewModel
this.allResearchProjects : IAllResearchProjects
TrySaveRe...
Try
[try]
Add
allResearchProjects.Add(Researc...
<<return>>
Return
return true
[catch (Exception)]
Return
return false

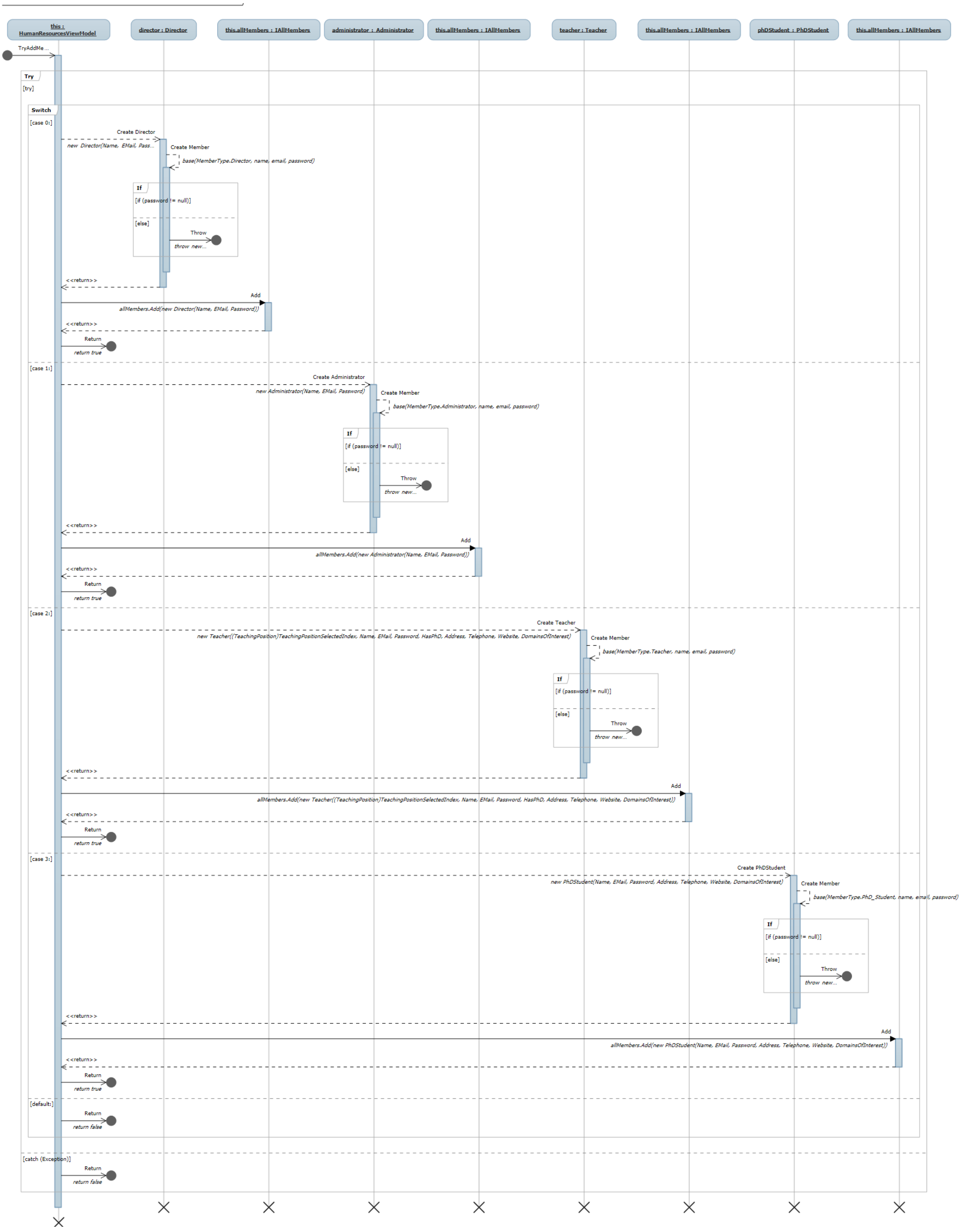

this : ResearchProjectViewModel
TryGetAllE...
Try
[try]
Return
return true
[catch (Exception)]
Return
return false

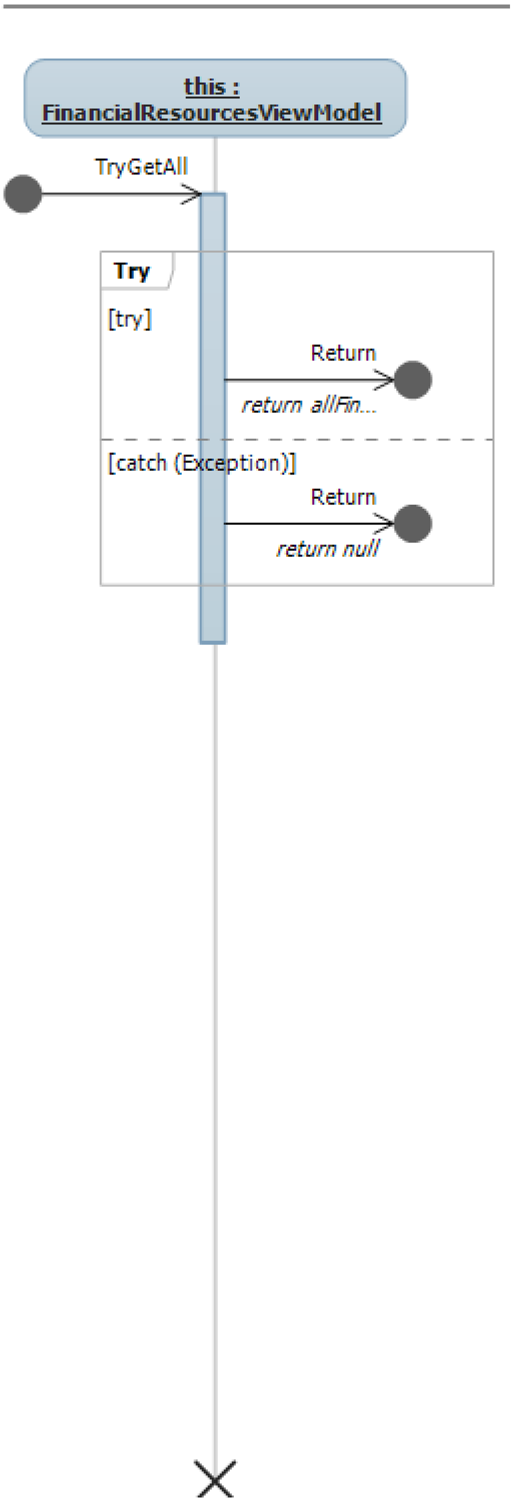

this : MemberViewModel
Logout

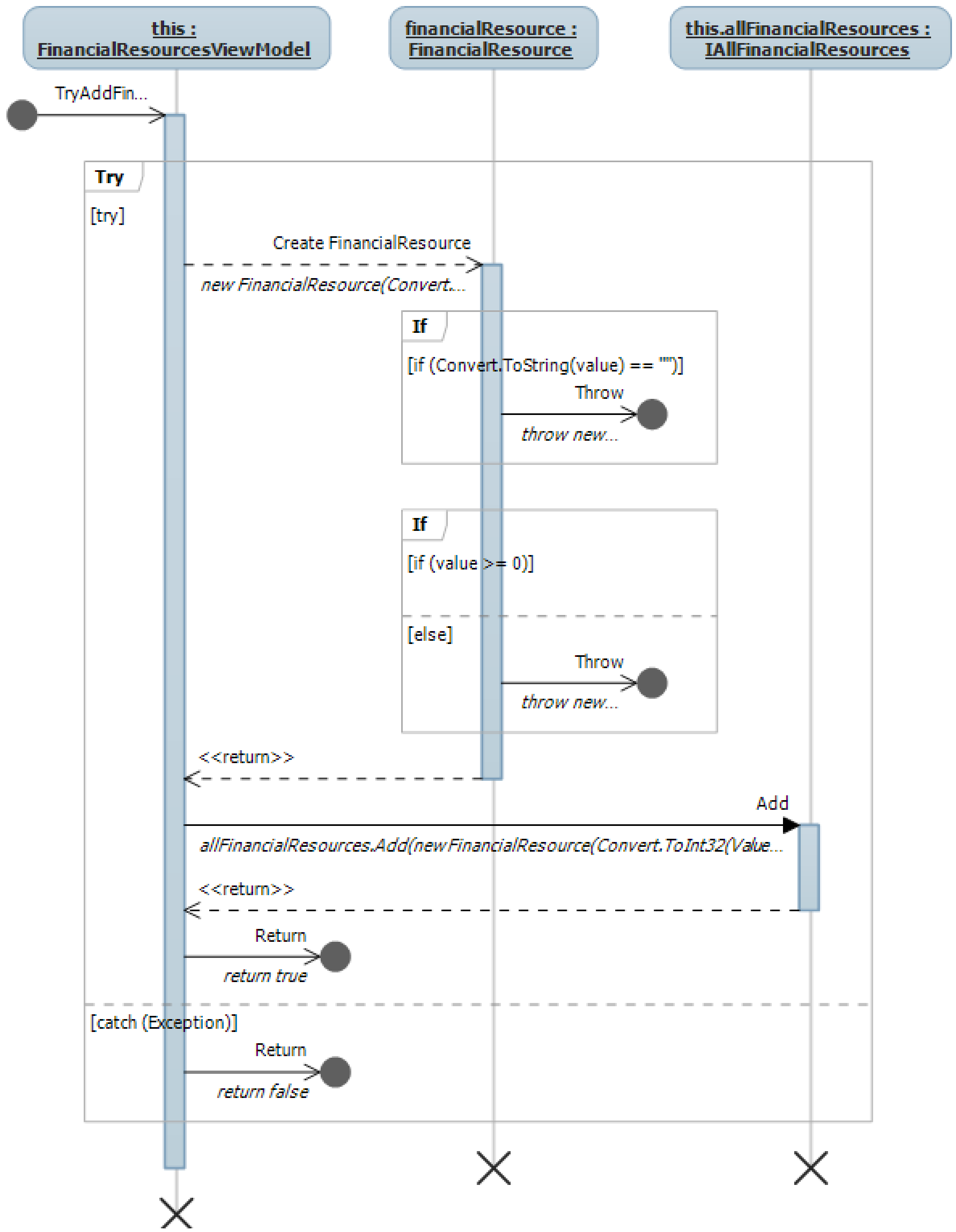

this : MemberViewModel
Logout

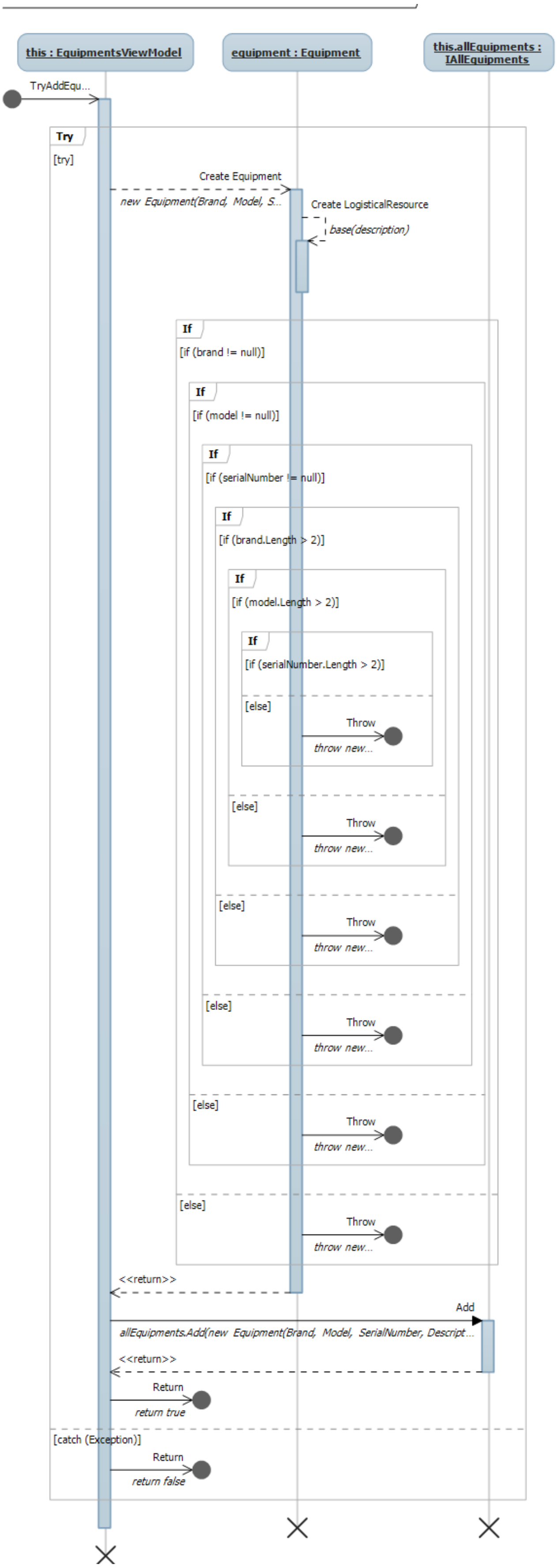

this : MemberViewModel
this.allMembers : IAllMembers
Authenticate
Where
allMembers.Where(email, passw...
<<return>>

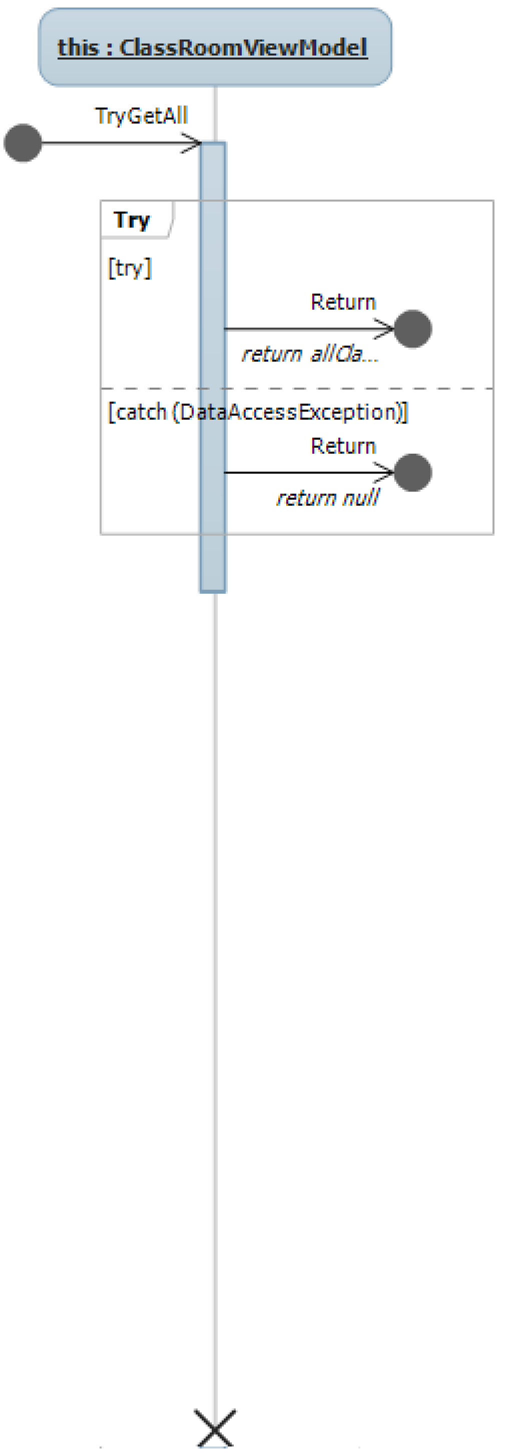

this : HumanResourcesViewModel
TryGetAll
Try
[try]
Return
return allMe...
[catch (Exception)]
Return
return null

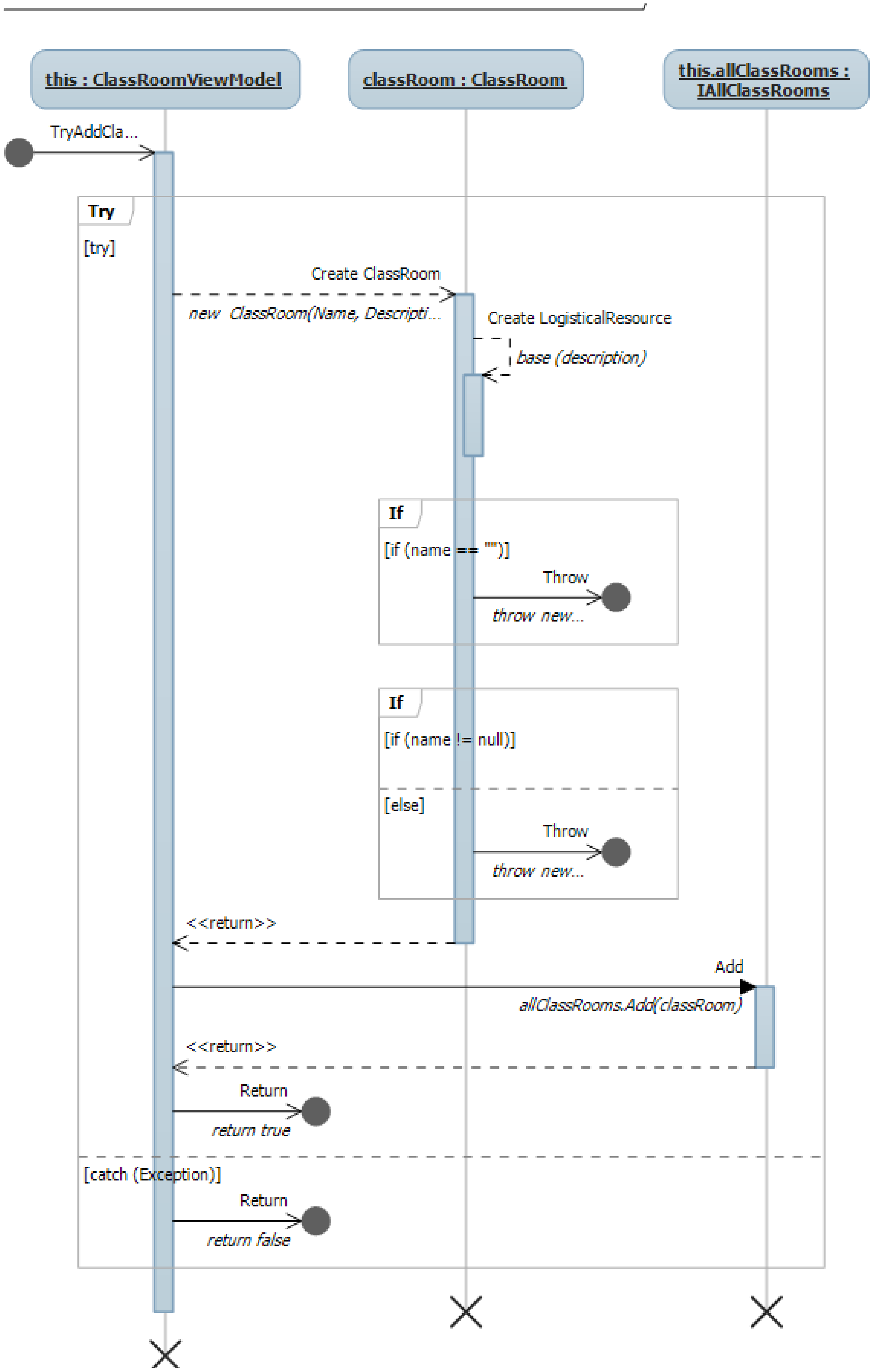

this : HumanResourcesViewModel
director : Director
this.allMembers : IAllMembers
administrator : Administrator
this.allMembers : IAllMembers
teacher : Teacher
this.allMembers : IAllMembers
phDStudent : PhDStudent
this.allMembers : IAllMembers
TryAddMe...
Try
[try]
Switch
[case 0:]
Create Director
new Director(Name, EMail, Pass...
Create Member
base(MemberType.Director, name, email, password)
If
[if (password != null)]
[else]
Throw
throw new...
<<return>>
Add
allMembers.Add(new Director(Name, EMail, Password))
<<return>>
Return
return true
[case 1:]
Create Administrator
new Administrator(Name, EMail, Password)
Create Member
base(MemberType.Administrator, name, email, password)
If
[if (password != null)]
[else]
Throw
throw new...
<<return>>
Add
allMembers.Add(new Administrator(Name, EMail, Password))
<<return>>
Return
return true
[case 2:]
Create Teacher
new Teacher((TeachingPosition)TeachingPositionSelectedIndex, Name, EMail, Password, HasPhD, Address, Telephone, Website, DomainsOfInterest)
Create Member
base(MemberType.Teacher, name, email, password)
If
[if (password != null)]
[else]
Throw
throw new...
<<return>>
Add
allMembers.Add(new Teacher((TeachingPosition)TeachingPositionSelectedIndex, Name, EMail, Password, HasPhD, Address, Telephone, Website, DomainsOfInterest))
<<return>>
Return
return true
[case 3:]
Create PhDStudent
new PhDStudent(Name, EMail, Password, Address, Telephone, Website, DomainsOfInterest)
Create Member
base(MemberType.PhD_Student, name, email, password)
If
[if (password != null)]
[else]
Throw
throw new...
<<return>>
Add
allMembers.Add(new PhDStudent(Name, EMail, Password, Address, Telephone, Website, DomainsOfInterest))
<<return>>
Return
return true
[default:]
Return
return false
[catch (Exception)]
Return
return false

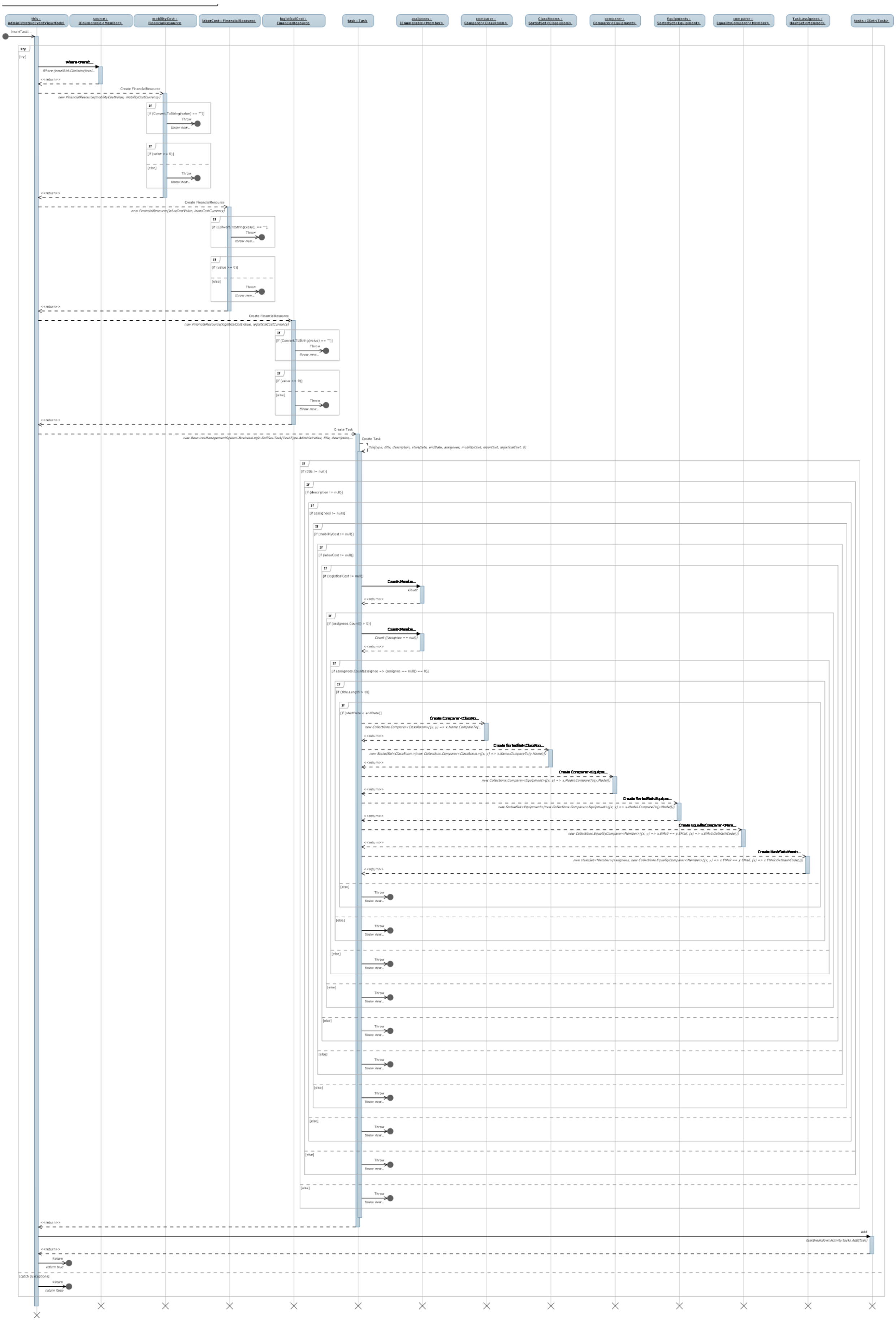

this : FinancialResourcesViewModel
TryGetAll
Try
[try]
Return
return allFin...
[catch (Exception)]
Return
return null

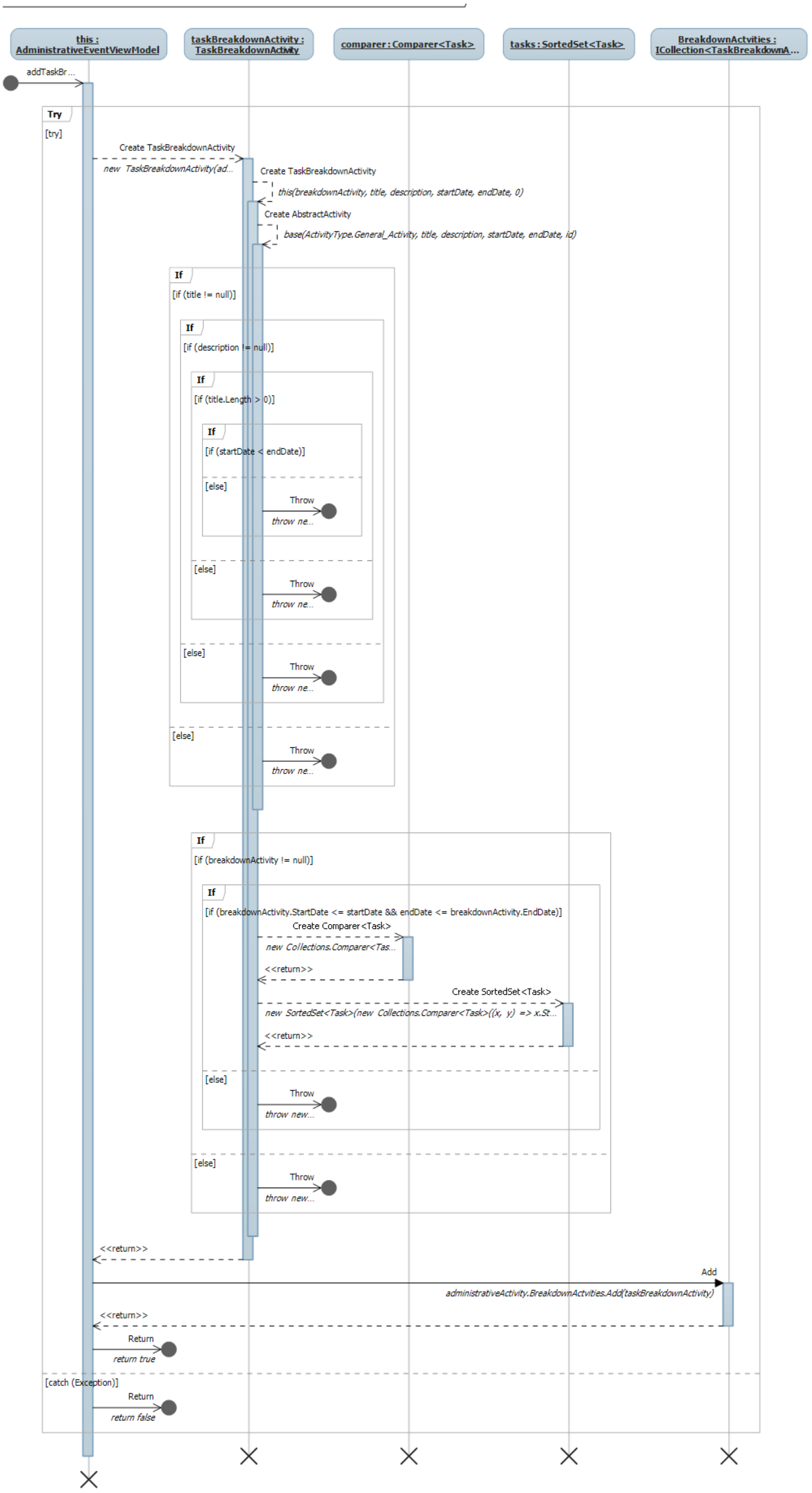

this : FinancialResourcesViewModel
financialResource : FinancialResource
this.allFinancialResources : IAllFinancialResources
TryAddFin...
Try
[try]
Create FinancialResource
new FinancialResource(Convert....
If
[if (Convert.ToString(value) == "")]
Throw
throw new...
If
[if (value >= 0)]
[else]
Throw
throw new...
<<return>>
Add
allFinancialResources.Add(new FinancialResource(Convert.ToInt32(Value...
<<return>>
Return
return true
[catch (Exception)]
Return
return false

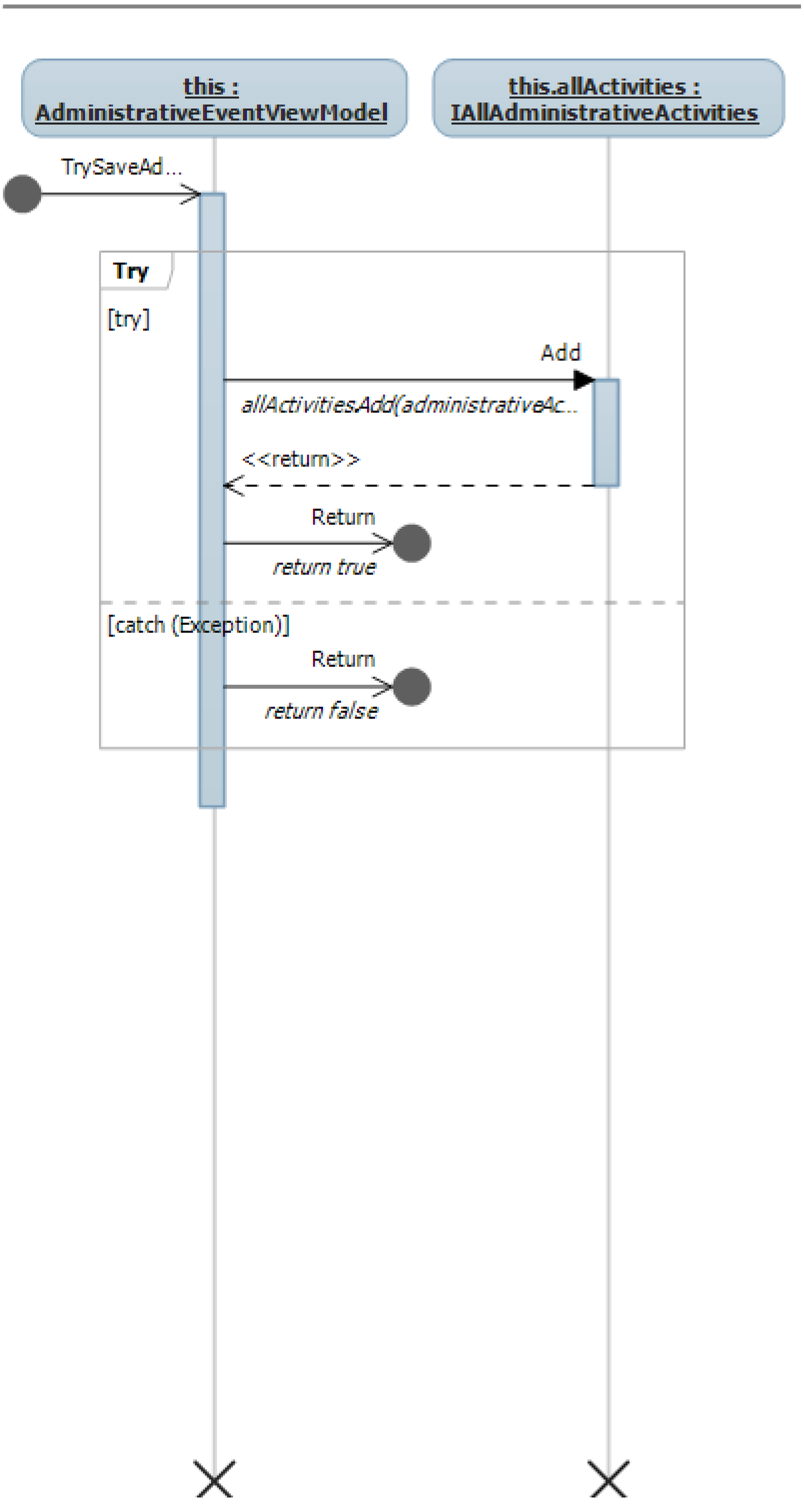

this : EquipmentsViewModel
equipment : Equipment
this.allEquipments : IAllEquipments
TryAddEqu...
Try
[try]
Create Equipment
new Equipment(Brand, Model, S...
Create LogisticalResource
base(description)
If
[if (brand != null)]
If
[if (model != null)]
If
[if (serialNumber != null)]
If
[if (brand.Length > 2)]
If
[if (model.Length > 2)]
If
[if (serialNumber.Length > 2)]
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
<<return>>
Add
allEquipments.Add(new Equipment(Brand, Model, SerialNumber, Descript...
<<return>>
Return
return true
[catch (Exception)]
Return
return false

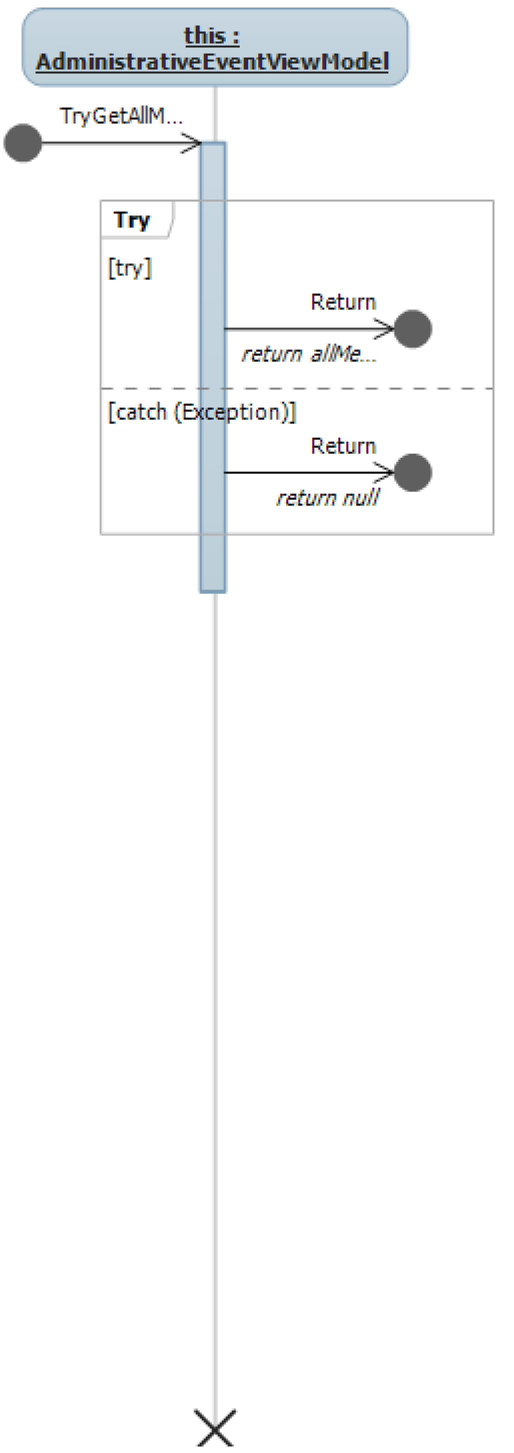

this : ClassRoomViewModel
TryGetAll
Try
[try]
Return
return allCla...
[catch (DataAccessException)]
Return
return null

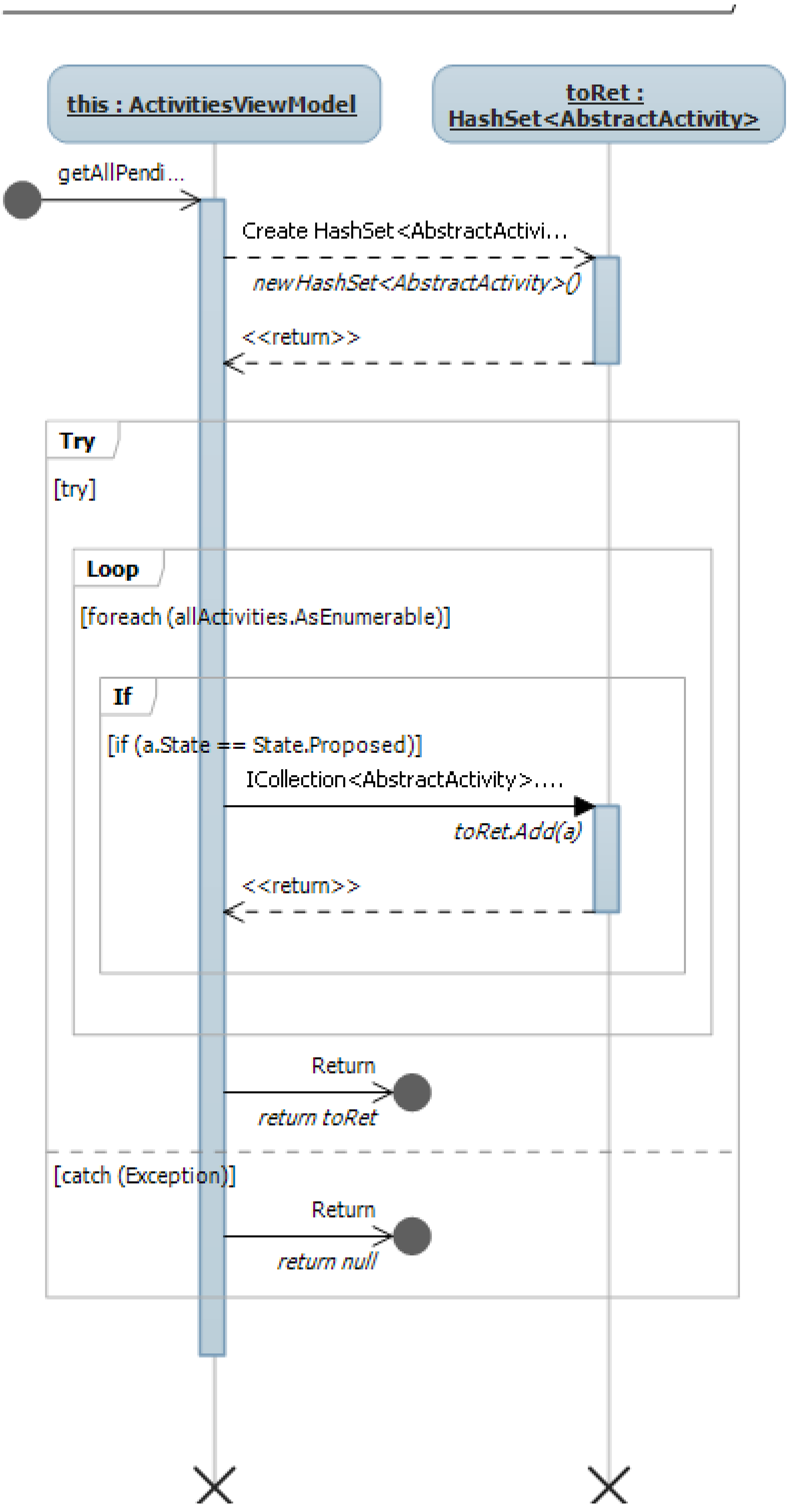

this : ClassRoomViewModel
classRoom : ClassRoom
this.allClassRooms : IAllClassRooms
TryAddCla...
Try
[try]
Create ClassRoom
new ClassRoom(Name, Descripti...
Create LogisticalResource
base (description)
If
[if (name == "")]
Throw
throw new...
If
[if (name != null)]
[else]
Throw
throw new...
<<return>>
Add
allClassRooms.Add(classRoom)
<<return>>
Return
return true
[catch (Exception)]
Return
return false

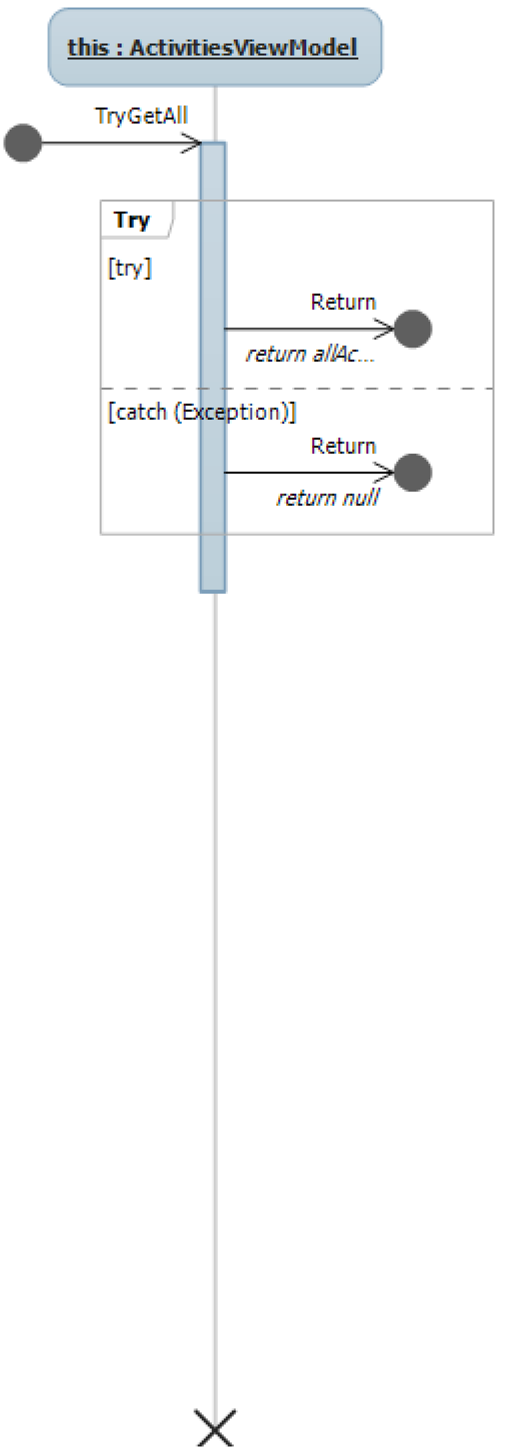

this : AdministrativeEventViewModel
source : IEnumerable<Member>
mobilityCost : FinancialResource
laborCost : FinancialResource
logisticalCost : FinancialResource
task : Task
assignees : IEnumerable<Member>
comparer : Comparer<ClassRoom>
ClassRooms : SortedSet<ClassRoom>
comparer : Comparer<Equipment>
Equipments : SortedSet<Equipment>
comparer : EqualityComparer<Member>
Task.assignees : HashSet<Member>
tasks : ISet<Task>
insertTask...
Try
[try]
Where<Memb...
Where (emailList.Contains(local...
<<return>>
Create FinancialResource
new FinancialResource(mobilityCostValue, mobilityCostCurrency)
If
[If (Convert.ToString(value) == "")]
Throw
throw new...
If
[If (value >= 0)]
[else]
Throw
throw new...
<<return>>
Create FinancialResource
new FinancialResource(laborCostValue, laborCostCurrency)
If
[If (Convert.ToString(value) == "")]
Throw
throw new...
If
[If (value >= 0)]
[else]
Throw
throw new...
<<return>>
Create FinancialResource
new FinancialResource(logisticalCostValue, logisticalCostCurrency)
If
[If (Convert.ToString(value) == "")]
Throw
throw new...
If
[If (value >= 0)]
[else]
Throw
throw new...
<<return>>
Create Task
new ResourceManagementSystem.BusinessLogic.Entities.Task(TaskType.Administrative, title, description,...
Create Task
this(type, title, description, startDate, endDate, assignees, mobilityCost, laborCost, logisticalCost, 0)
If
[If (title != null)]
If
[If (description != null)]
If
[If (assignees != null)]
If
[If (mobilityCost != null)]
If
[If (laborCost != null)]
If
[If (logisticalCost != null)]
Count<Memb...
Count
<<return>>
If
[If (assignees.Count() > 0)]
Count<Memb...
Count ((assignee == null))
<<return>>
If
[If (assignees.Count(assignee => (assignee == null)) == 0)]
If
[If (title.Length > 0)]
If
[If (startDate < endDate)]
Create Comparer<ClassRo...
new Collections.Comparer<ClassRoom>((x, y) => x.Name.CompareTo(...
<<return>>
Create SortedSet<ClassRo...
new SortedSet<ClassRoom>(new Collections.Comparer<ClassRoom>((x, y) => x.Name.CompareTo(y.Name)))
<<return>>
Create Comparer<Equipm...
new Collections.Comparer<Equipment>((x, y) => x.Model.CompareTo(y.Model))
<<return>>
Create SortedSet<Equipm...
new SortedSet<Equipment>(new Collections.Comparer<Equipment>((x, y) => x.Model.CompareTo(y.Model)))
<<return>>
Create EqualityComparer<Mem...
new Collections.EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) => x.EMail.GetHashCode())
<<return>>
Create HashSet<Memb...
new HashSet<Member>(assignees, new Collections.EqualityComparer<Member>((x, y) => x.EMail == y.EMail, (x) => x.EMail.GetHashCode()))
<<return>>
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
[else]
Throw
throw new...
<<return>>
Add
taskBreakdownActivity.tasks.Add(task)
<<return>>
Return
return true
[catch (Exception)]
Return
return false

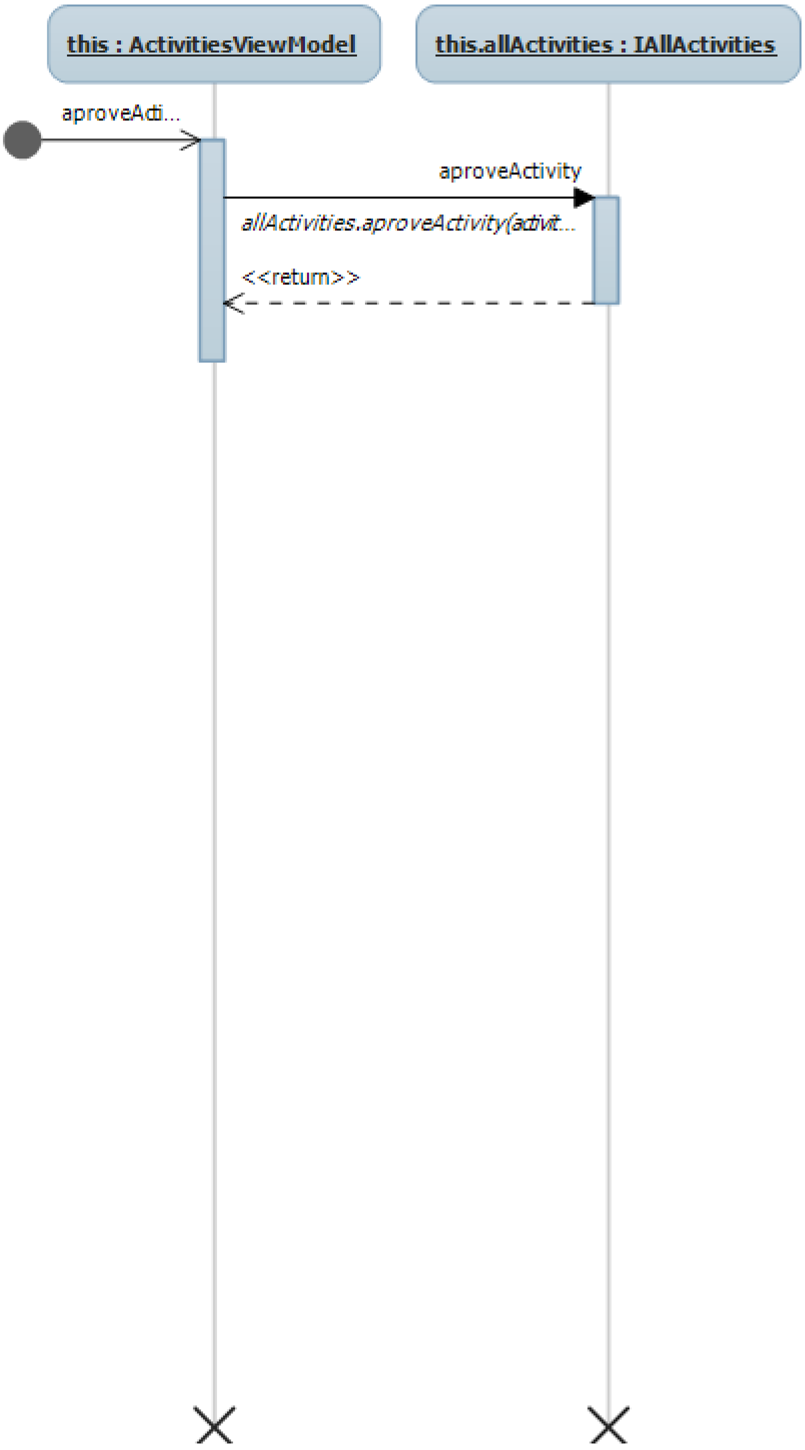

this : AdministrativeEventViewModel
taskBreakdownActivity : TaskBreakdownActivity
comparer : Comparer<Task>
tasks : SortedSet<Task>
BreakdownActvities : ICollection<TaskBreakdownA...
addTaskBr...
Try
[try]
Create TaskBreakdownActivity
new TaskBreakdownActivity(ad...
Create TaskBreakdownActivity
this(breakdownActivity, title, description, startDate, endDate, 0)
Create AbstractActivity
base(ActivityType.General_Activity, title, description, startDate, endDate, id)
If
[if (title != null)]
If
[if (description != null)]
If
[if (title.Length > 0)]
If
[if (startDate < endDate)]
[else]
Throw
throw ne...
[else]
Throw
throw ne...
[else]
Throw
throw ne...
[else]
Throw
throw ne...
If
[if (breakdownActivity != null)]
If
[if (breakdownActivity.StartDate <= startDate && endDate <= breakdownActivity.EndDate)]
Create Comparer<Task>
new Collections.Comparer<Tas...
<<return>>
Create SortedSet<Task>
new SortedSet<Task>(new Collections.Comparer<Task>((x, y) => x.St...
<<return>>
[else]
Throw
throw new...
[else]
Throw
throw new...
<<return>>
Add
administrativeActivity.BreakdownActvities.Add(taskBreakdownActivity)
<<return>>
Return
return true
[catch (Exception)]
Return
return false

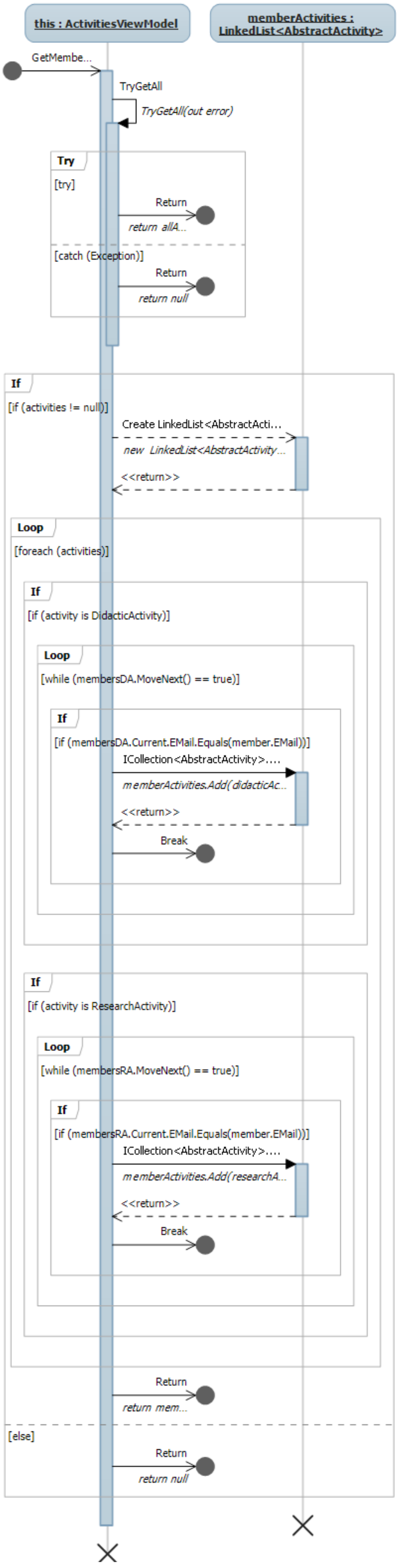

this : AdministrativeEventViewModel
this.allActivities : IAllAdministrativeActivities
TrySaveAd...
Try
[try]
Add
allActivities.Add(administrativeAc...
<<return>>
Return
return true
[catch (Exception)]
Return
return false


this : AdministrativeEventViewModel
TryGetAllM...
Try
[try]
Return
return allMe...
[catch (Exception)]
Return
return null


this : ActivitiesViewModel
toRet : HashSet<AbstractActivity>
getAllPendi...
Create HashSet<AbstractActivi...
newHashSet<AbstractActivity>()
<<return>>
Try
[try]
Loop
[foreach (allActivities.AsEnumerable)]
If
[if (a.State == State.Proposed)]
ICollection<AbstractActivity>....
toRet.Add(a)
<<return>>
Return
return toRet
[catch (Exception)]
Return
return null


this : ActivitiesViewModel
TryGetAll
Try
[try]
Return
return allAc...
[catch (Exception)]
Return
return null


this : ActivitiesViewModel
this.allActivities : IAllActivities
aproveActi...
aproveActivity
allActivities.aproveActivity(activit...
<<return>>


this : ActivitiesViewModel
memberActivities : LinkedList<AbstractActivity>
GetMembe...
TryGetAll
TryGetAll(out error)
Try
[try]
Return
return allA...
[catch (Exception)]
Return
return null
If
[if (activities != null)]
Create LinkedList<AbstractActi...
new LinkedList<AbstractActivity...
<<return>>
Loop
[foreach (activities)]
If
[if (activity is DidacticActivity)]
Loop
[while (membersDA.MoveNext() == true)]
If
[if (membersDA.Current.EMail.Equals(member.EMail))]
ICollection<AbstractActivity>....
memberActivities.Add(didacticAc...
<<return>>
Break
If
[if (activity is ResearchActivity)]
Loop
[while (membersRA.MoveNext() == true)]
If
[if (membersRA.Current.EMail.Equals(member.EMail))]
ICollection<AbstractActivity>....
memberActivities.Add(researchA...
<<return>>
Break
Return
return mem...
[else]
Return
return null

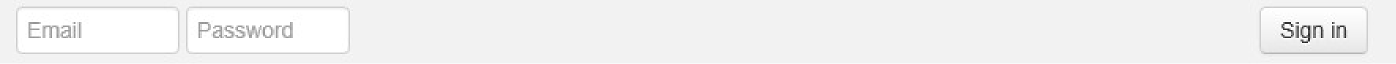
User Manual

1) Introduction

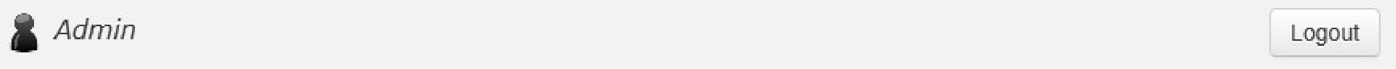
This section is dedicated to the user manual. Here one can find details about how to use the application.

2) User authentication

When accessing the application via a web browser, on the top side there is an authentication form:

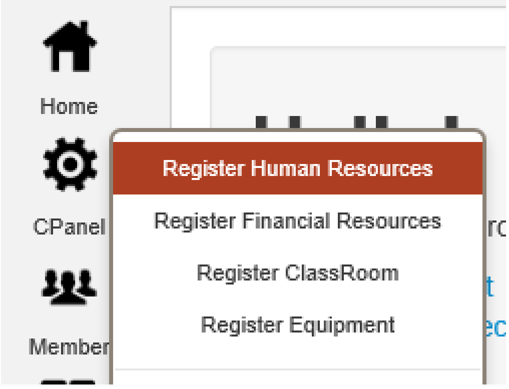


To authenticate simply write your E-Mail and password. After that either hit enter or click on the "Sign in" button. If the authentication fails the form will be left empty. If the authentication succeeds the form will be replaced by a similar view:

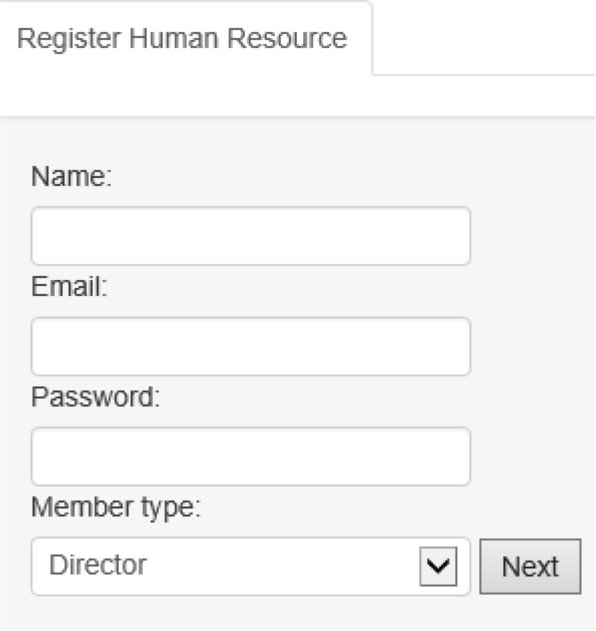


3) Registering a human resource

To be able to add human resources you must be authenticated as an administrator. After you have successfully logged in as an admin, go on the left side panel and select the register human resource button from the menu.



After clicking on the Register Human Resources item, you will be asked to fill a form such as this:

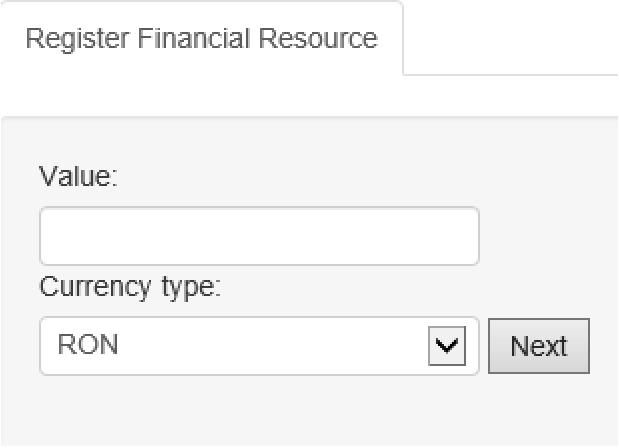


Member type: Director

Simply complete the form and follow the onscreen details to register the human resource.

4) Registering a financial resource

Similar to feature 3, simply navigate to the left menu, after successfully authenticating as an administrator, and select the "Register Financial Resources" option. A form similar to this will appear:

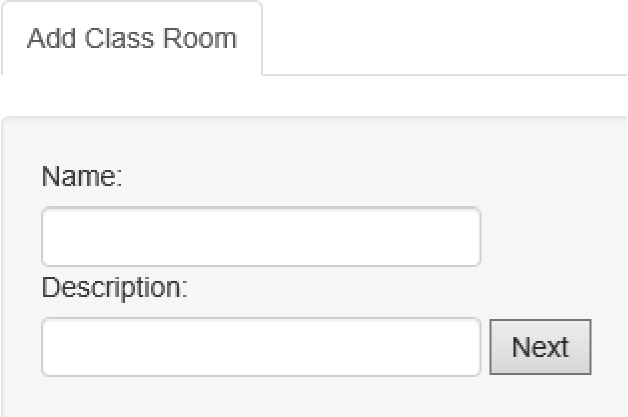


Currency type: RON

Complete the form and click next, follow the onscreen details to complete the registration.

5) Registering a class room

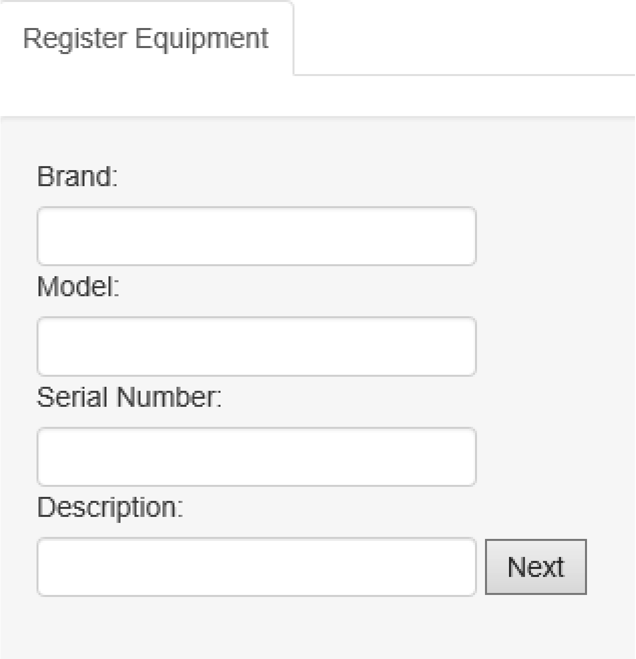
Similar to feature 3, simply navigate to the left menu, after successfully authenticating as an admin, and select the "Register Class Room" option. A form similar to this will appear:



Complete the form and click next, follow the onscreen details to complete the registration.

6) Registering equipment

Similar to feature 3, simply navigate to the left menu, after successfully authenticating as an admin, and select the "Register Equipment" option. A form similar to this will appear:



Register Equipment

Brand:

Model:

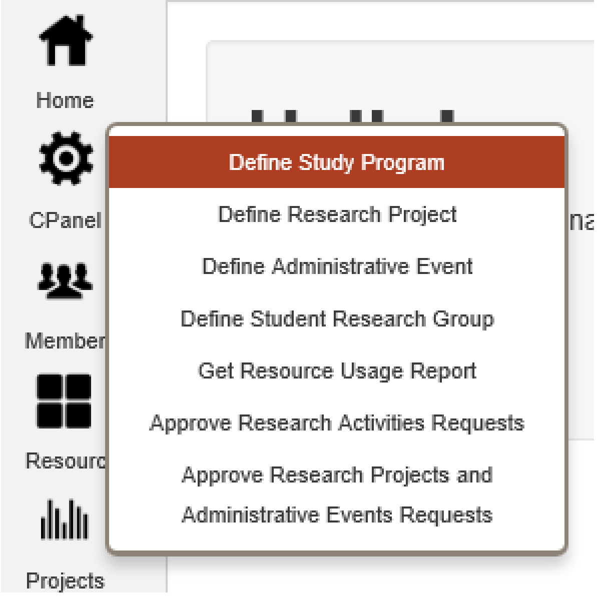
Serial Number:

Description:

Complete the form and click next, follow the onscreen details to complete the registration.

7) Defining study programs

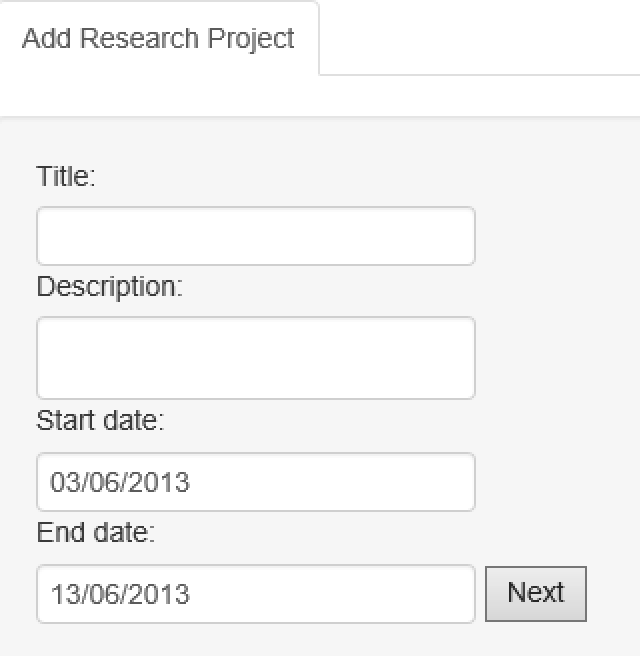
To add a study program you must authenticate as a director. After doing so you can access this feature by navigating through the left menu:


Home
CPanel
Member
Resourc
Projects
Define Study Program
Define Research Project
Define Administrative Event
Define Student Research Group
Get Resource Usage Report
Approve Research Activities Requests
Approve Research Projects and Administrative Events Requests

After clicking on that menu item you will be given a form to complete (has a lot of fields). After completing each field simply click next and follow the onscreen details to know whether the operation succeeded or not and what to do in case it's the former.

8) Defining research projects

Just like in feature 7, navigate through the left menu and click on the "Define Research Project" item and a form like this will appear:



Add Research Project

Title:

Description:

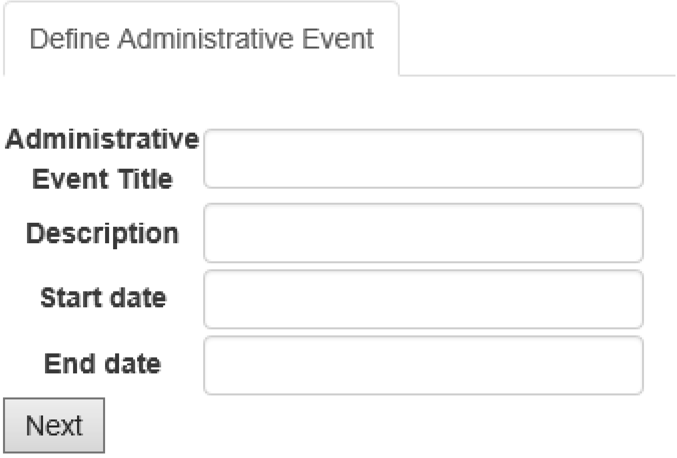
Start date: 03/06/2013

End date: 13/06/2013

Complete the form and click next, follow the onscreen details to complete the registration.

9) Defining administrative activities

Just like in feature 7, navigate through the left menu and click on the "Define Administrative Event" item and a form like this will appear:



Define Administrative Event

Administrative Event Title

Description

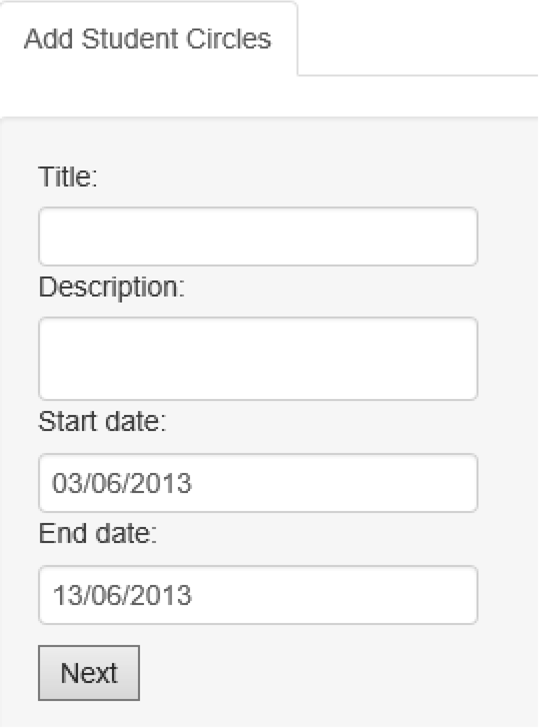
Start date

End date

Complete the form and click next, follow the onscreen details to complete the registration.

10) Defining Student Research Group (student circle)

Just like in feature 7, navigate through the left menu and click on the “Define Student Research Group” item and a form like this will appear:



Add Student Circles

Title:

Description:

Start date: 03/06/2013

End date: 13/06/2013

Complete the form and click next, follow the onscreen details to complete the registration.

11) Get resource usage report

Just like in feature 7, navigate through the left menu and click on the “Get Resource Usage Report” item and the application will generate the requested report and display it.

12) Approve Research Activities Requests

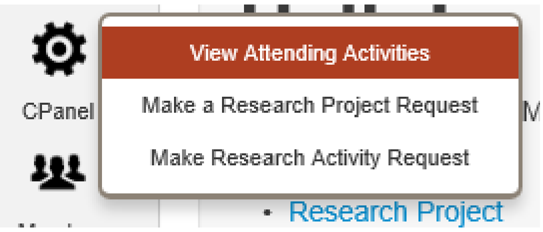
Just like in feature 7, navigate through the left menu and click on the “Approve Research Activities Requests” item and the application will display a list of pending activities. Click on any of the activities to be able to approve it.

13) Approve Research Projects and Administrative Events Requests

Just like in feature 7, navigate through the left menu and click on the “Approve Research Projects and Administrative Events Requests” item and the application will display a list of pending research projects and administrative events. Click on any of the activities to be able to approve it.

14) Obtaining attending activities calendar

To add a study program you must authenticate as a teacher. After doing so you can access this feature by navigating through the left menu:


CPanel
View Attending Activities
Make a Research Project Request
Make Research Activity Request
Research Project

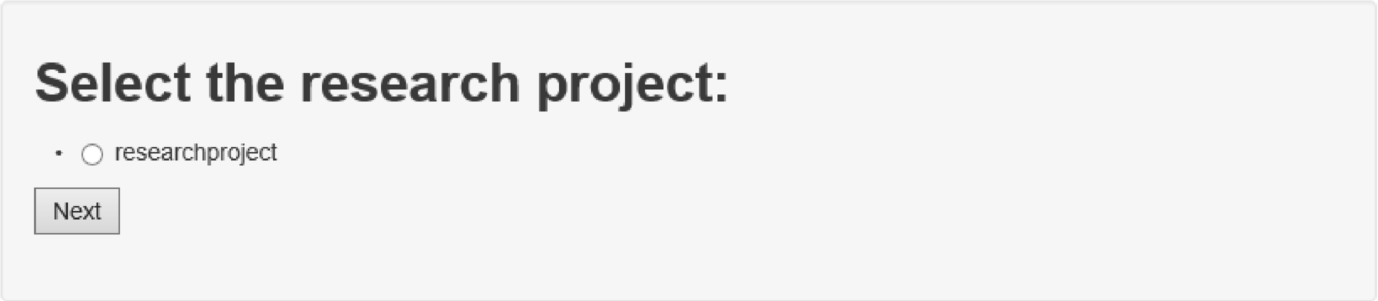
After clicking on “View Attending Activities” the application will display a table with all activities the teacher is attending to.

15) Make Research Project Request

Just like feature 14, navigate to the left menu and click on the “Make a Research Project Request” item. The displayed form is similar to the one in feature 8. Complete the form and follow the onscreen details to request a research project.

16) Make Research Activity Request

Just like feature 14, navigate to the left menu and click on the “Make Research Activity Request” item. The application will display a form like this:



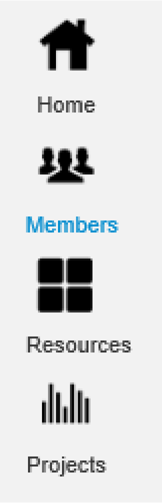
Select the research project:

- ○ researchproject

Complete the form and click next, follow the onscreen details to complete the registration.

17) Obtaining general information about resources and project (available to public)

To obtain public information you do not require an authentication. Simply navigate through the left panel:



From here you can view a list of members currently registered, a list of logistical resources currently owned and finally a list of projects that the department has.