Web-Based Evaluations

Technical Manual

Class CPSC 488 Section 02

Group Number: 2

Revised: 4/22/2023

Originally Created: 4/25/2022

Tanuj Rane [txr1029@sru.edu](mailto:txr1029@sru.edu), ranetanuj@gmail.com

Dalton Stenzel [drs1030@sru.edu](mailto:drs1030@sru.edu), daltonrstenzel@gmail.com

Logan Racer [lsr1006@sru.edu](mailto:lsr1006@sru.edu), loganracer27@gmail.com

Anthony Cinicola [ajc1033@sru.edu](mailto:ajc1033@sru.edu)

Duncan Lawrence [dal1017@sru.edu](mailto:dal1017@sru.edu) duncanlawrence2000@gmail.com

J Abbigail Rowe [jhr1002@sru.edu](mailto:jhr1002@sru.edu) jakerowe@protonmail.com

Zachary Freilino zlf1001@sru.edu, zfreilino@gmail.com

David Gillette dgg1001@sru.edu, davidgsgillette@gmail.com

Revised By:

Chad Green [cwg1003@sru.edu](mailto:cwg1003@sru.edu)

Contents

[Structure of the Program 5](#_Toc146707059)

[Thymeleaf and Method Associations 5](#_Toc146707060)

[Company Structure 5](#_Toc146707061)

[Classes of Interest 6](#_Toc146707062)

[Class Relationships 8](#_Toc146707063)

[UserController & UserService 8](#_Toc146707064)

[UserController & AdminMethodsService 8](#_Toc146707065)

[DataVisualizationController & CreateDataset 8](#_Toc146707066)

[EvalFormController, ParseEvaluation, & GenerateEvalReport 9](#_Toc146707067)

[Evaluation, Section, Question, and ComputeRange 9](#_Toc146707068)

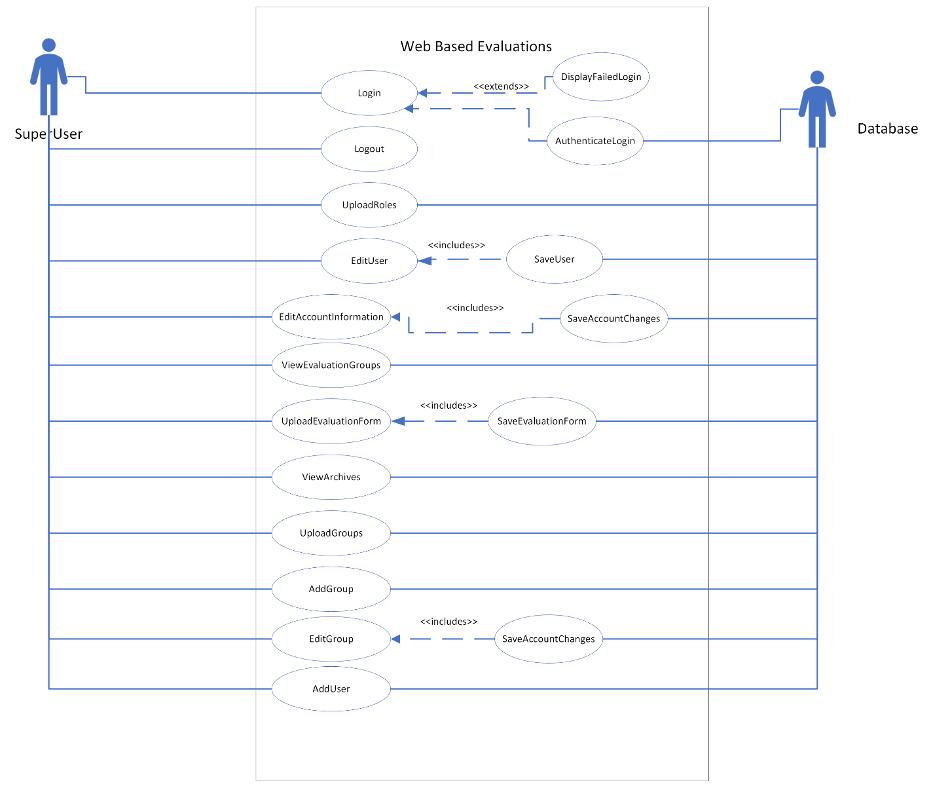
[Applications Properties 9](#_Toc146707069)

[Caveat 10](#_Toc146707070)

[UML Diagrams for the System 11](#_Toc146707071)

[Use Case of the Evaluator Role 11](#_Toc146707072)

[Use Case of the SuperUser/Admin Role 12](#_Toc146707073)

[ 12](#_Toc146707074)

[Use Case of the SuperSuperUser Role 13](#_Toc146707075)

[Use Case of the Evaluator Admin 14](#_Toc146707076)

[Sequence Diagrams 15](#_Toc146707077)

[CompanySuperUser on the Manage Users Page 15](#_Toc146707078)

[CompanySuperUser on the Manage Roles Page 15](#_Toc146707079)

[Evaluator Edit Account Information 16](#_Toc146707080)

[Evaluator Perform Evaluation 16](#_Toc146707081)

[Uploading Evaluation Forms 17](#_Toc146707082)

[Uploading Groups 18](#_Toc146707083)

[Uploading Companies 19](#_Toc146707084)

Caveat 8

# Structure of the Program

## Thymeleaf and Method Associations

This program uses Spring Boot and Maven dependencies. Part of these dependencies includes Thymeleaf, which allows HTML to connect to Java code. In the controller classes (i.e., those in the controller package), each of the return values for the methods correspond to an html file—in other words, when the method completes, it will then refer to a specific html file in the src/main/resources/templates folder. In the html files, there are tokens that appear like “@{/home}.” These are linking to other pages in the website, and thus, referring to other methods. This means, when a certain button is pushed, go to the page mentioned in the token. To change what happens when a button is pushed, you need to edit the method called by that button.

## Company Structure

This program also features what we have referred to as a “company structure”. This boils down to how the tables are structured in the database, as a tree. The root of the tree is the World, which then has many Continents, each of which has Countries, then Provinces, then Cities, then finally Locations. Each Company consists of departments which are collections of users, and location groups, which are (as the name suggests) groups of locations. Each department has one user which is the department head.

# Classes of Interest

**AddDepartmentController.java** - Class for housing the methods for controlling how to add departments manually.

**AddRolesController.java** - Class for housing the methods for controlling how to add roles via uploading.

**AddUserController.java -** Class for housing the methods for controlling how to add users manually or via uploading.

**AdminMethodsService.java -** This class provides all the tools for the UserController and AddUserController classes that don't require the use of a html file/link tag.

**Archive.java -** Class for methods of an Archive object, made of getters and setters. Represents deleted evaluation.

**ArchiveController.java -** Class for handling Archive-related changes to groups like adding, retrieving, and displaying information.

**ComputeRange.java -** Contains a range of min and max values which correspond to a given range name. Used in an Evaluation for determining what grade to give for a score.

**CompanyController.java** - Class for housing the methods for controlling how to add companies via uploading as well as editing companies.

**CreateDataset** - Class for creating and loading all datasets to be used in chart generation and Pdf reports.

**DataVisualizationController.java -** Class forhandling the mapping and generation of charts (pie chart, bar chart, area chart, ring chart) as well as the group Pdf reports.

**EvalFormController.java -** Controller for functionality of the 'eval\_templates.html' web page for 'ADMIN\_EVAL' users. Includes uploading, saving, deleting, and downloading excel files and Evaluation forms.

**Evalrole.java** - Class for methods of an Evalrole object, made of getters and setters. Represents roles that an evaluator is assigned to.

**Evaluation.java -** Contains all data, methods, and formatting information relating to an Evaluation. Can be serialized for storage in a database.

**EvaluationLog.java** - Class for methods of an Evaluationlog object, made of getters and setters. represent evaluation made by an evaluator on a reviewee.

**Evaluator.java** - Class for methods of an evaluator object, made of getters and setters. represents the evaluator in the group.

**EvaluatorController.java -** Class for handling Evaluator-related changes to groups like adding editing, deleting, retrieving, and displaying information.

**GenerateEvalReport.java -** Contains methods for generating an Excel file based on a set of completed Evaluations.

**Group.java** - Class for methods of a group object, made of getters and setters. Stores lists of evaluators and reviewees.

**GroupController.java -** Class for handling Group related changes to groups like adding editing, deleting, retrieving, and displaying information.

**HomePage.java -** Class for controlling the starting sequence of users logging and taking them to the home page.

**ParseEvaluation.java -** Contains the methods for retrieving Evaluation data from an XML file.

**Privilege.java** - This class holds the specific privileges that can be given to a role. They include read, write, delete (RWD) and what group they have this power over.

**Question.java -** Contains all data, methods, and formatting information of a Question within an Evaluation. Can be serialized for storage in a database.

**ResetPasswordController.java -** Class for controlling attempts at password resets such as recovery and first-time logins.

**Reviewee.java** - Class for methods of a reviewee object, made of getters and setters. represents the individual being evaluated.

**RevieweeController. java -** Class for handling reviewee-related changes to groups like adding editing, deleting, retrieving, and displaying information.

**Role.java** - Class for methods of a role object, made of getters and setters. Stores the list of roles that will be given to users.

**Section.java -** Contains all data, methods, and formatting information of a Section within an Evaluation. Can be serialized for storage in a database.

**SecurityConfiguration.java** -Class for the security configuration where hash checks are performed as well as authority access to web pages/mappings.

**SelfEvaluation.java** - Class for methods of a self-evaluation object, made of getters and setters. represent evaluation made by a reviewee on themselves (note only if groups allow evaluation)

**User.java -** Class for methods of a user object, exclusively made of getters and setters.

**UserController.java -** Class for handling user related changes other than adding users.

**UserService.java -** Class for user lists and the sorting for them as well.

# Class Relationships

## UserController & UserService

* UserController uses UserServices’ various list-related methods to return a list that matches the description of the UserController’s parameters.
* UserService is usually called by the UserController to retrieve all users or to retrieve users relating to a keyword

## UserController & AdminMethodsService

* AdminMethodsService plays a role in holding a wide variety of methods used mostly for the UserController, such as a pageCalc() method for figuring out how to display the users given a series of parameters relating to a search such as users per page, and how many pages should the list of users be divided across. The pageCalc() method will return a list that the UserController will then display in the users table.
* AdminMethodsService also holds the adminUserPageItems() method as well to add all the attributes that the current method is holding to the webpage. An example of an attribute would be the current page the user is on for viewing the users in the table. Nothing is returned from this method.
* Another method the UserController will call from the AdminMethodsService is the comparingMethod() that sends two users to be compared for differences and invalid inputs/changes. It consists of a series of checks for each attribute of the users. If any acceptable changes occur, a response message will display, but if there are also invalid changes present then only the valid ones will apply.

## DataVisualizationController & CreateDataset

* DataVisualizationController is where the charts are generated and sent to be displayed on either the web page in the case of individual evaluation charts, or in the form of a Pdf report for the whole group.
* To pull the data stored in the evaluation forms the DataVisualizationController calls different methods in the createDataset class. These dataset creation classes accept deserialized evaluation logs and use methods from the Evaluation.java class to access information in the evaluation. These include the section names, and section scores.

## EvalFormController, ParseEvaluation, & GenerateEvalReport

* The ParseEvaluation and GenerateEvalReport classes are accessed exclusively from within methods of the EvalFormController class.

## Evaluation, Section, Question, and ComputeRange

* These four classes work in conjunction to create an evaluation. Evaluation contains a list of Sections and Section, contains a list of Questions. Both Evaluation and Question contain a list of ComputeRanges.

# Applications Properties

**Spring.servlet.multipart.enabled** – Allows for spring to break up “large” files into smaller chunks to be uploaded

**Spring.servlet.multipart.file-size-threshold** – Sets the maximum value before files are written to disk.

**Spring.servlet.multipart.max-file-size** – Sets the maximum size a file may be uploaded to the program. Does not equate to the maximum file size that the mySQL server may handle.

**Spring.servlet.multipart.max-request-size** – Sets the maximum size the entire upload. A set of files cannot exceed the set value. Does not equate to the maximum upload size that the mySQL server may handle.

**Spring.datasource.url** – The URL, otherwise known as a link, to the server location to access data from said server. Additionally, the URL contains the ability to automatically create the schema rather than importing it from a .SQL file.

**Spring.datasource.username** – The login username associated with the server attempting to be accessed.

**Spring.datasource.password** – The login password associated with the server attempting to be accessed.

**Spring.jpa.hibernate.ddl-auto** – The setting for JPA repositories which manages the handling of an embedded schema. Being set to none notes that the schema used is not embedded.

**Spring.datasource.initialization-mode** –The setting for handling how to manage the initialization of the database. The always option notes that the datasource will always be initialized.

**Spring.mail.host** – The email service to be used for sending emails.

**Spring.mail.port** – The port number of the email service selected.

**Spring.mail.username** – The username/email address that will be the address for sending emails.

**Spring.mail.password** – The password of the username/email address account.

**Spring.mail.properties.mail.smtp.auth** – Notes where there is authorization to send credentials or information to a mail server

**Spring.mail.properties.mail.smtp.starttls.enable** – Deals with whether the server requires a TLS (transport layer security) based connection. Some SMTP servers require it.

**Logging.path** – The location where the log file will be stored. The location is with respect to the root of the application, and not the operating system.

**Logging.file.name** – The name of the log file.

**Logginglevel.com.appsdevloperblog** – Denotes the type of logging used for the application

**Logging.level.root** – Determines the sets and types of errors able to be logged

# Caveat

The error was a cascading type error on association within group and review class and associated other classes in some cases that were slowing down the saving data in the database because of the dependency issue.

Here is the following solution and explanation about it.

Cascade Type: Check the cascade type configured for your association. If you have configured cascade deletion (e.g., CascadeType.REMOVE) on the association, updating a record might inadvertently trigger the deletion of related records. This behavior is expected because the cascading behavior propagates to the related records.

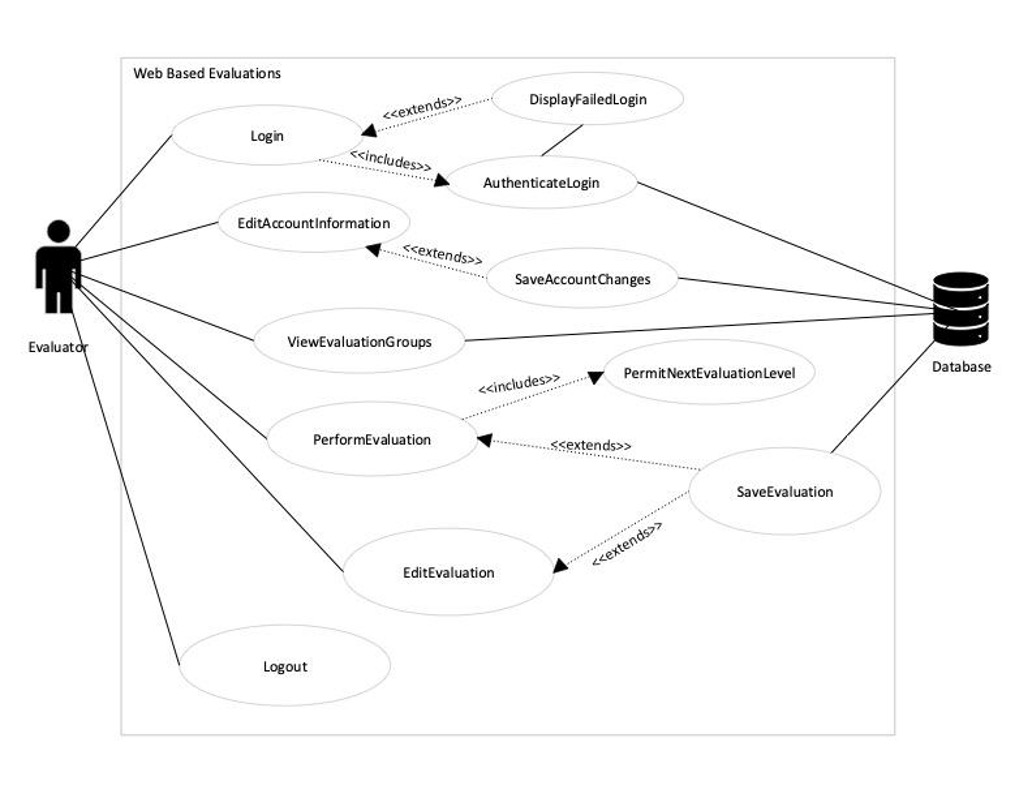
Solution: Review and adjust the cascade type as needed. If you don't want related records to be deleted when updating the main record, remove the CascadeType.REMOVE from the cascade configuration.

Bidirectional Relationships: In a bidirectional relationship (e.g., one-to-many with @OneToMany and @ManyToOne), ensure you update both sides of the relationship correctly. Please update both sides to avoid inconsistencies and unexpected behavior.

Solution: When modifying a bidirectional relationship, update both sides to maintain consistency.

# UML Diagrams for the System

## Use Case of the Evaluator Role



## Use Case of the SuperUser/Admin Role

# 

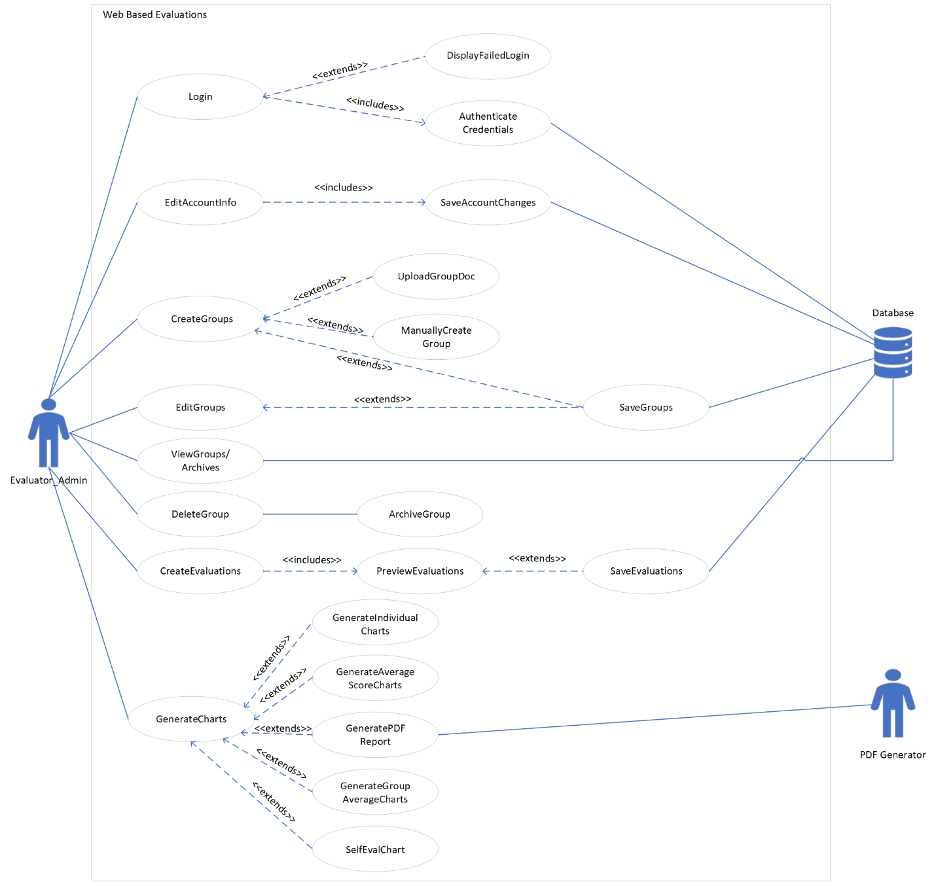
## 

## 

## Use Case of the SuperSuperUser Role

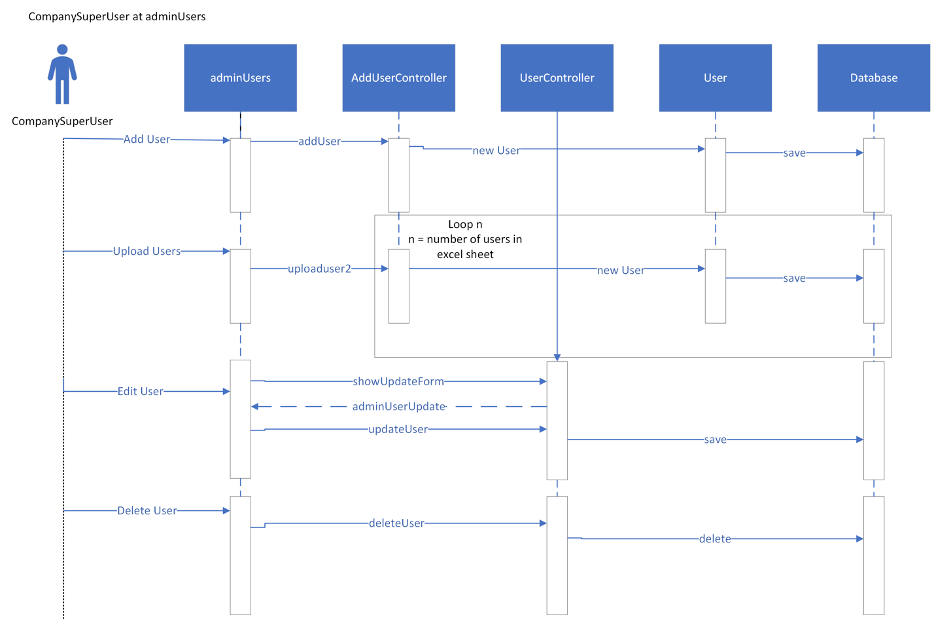


## Use Case of the Evaluator Admin

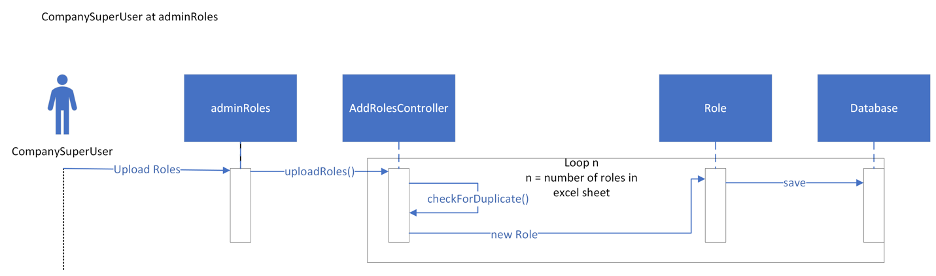


# Sequence Diagrams

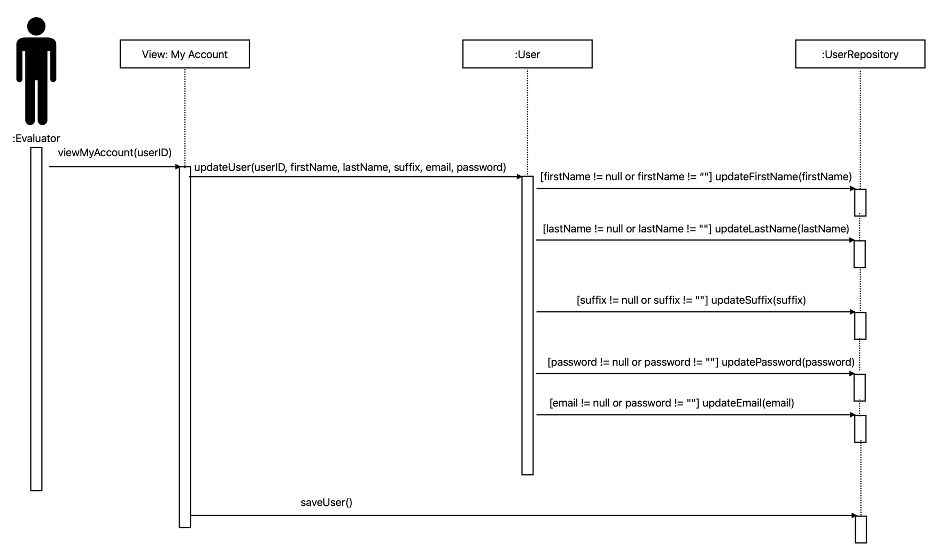
## CompanySuperUser on the Manage Users Page



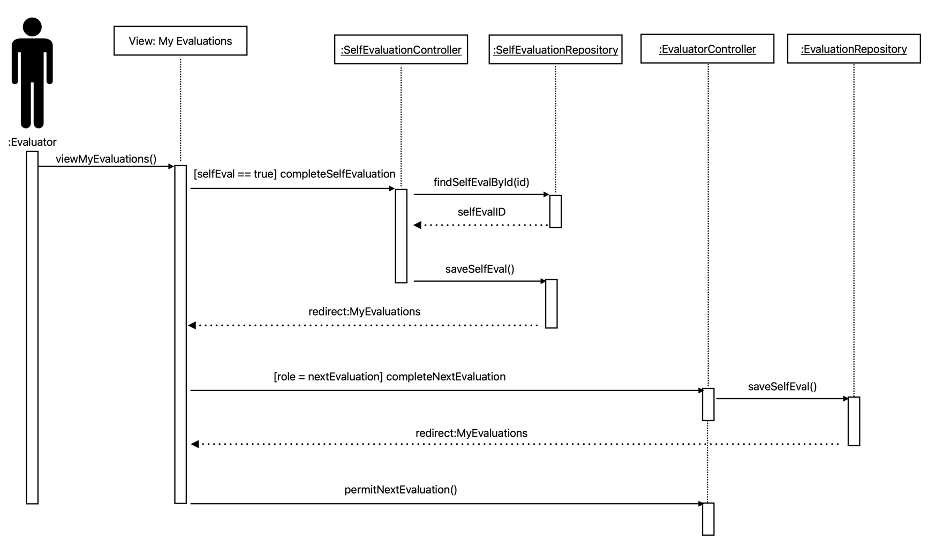
## CompanySuperUser on the Manage Roles Page



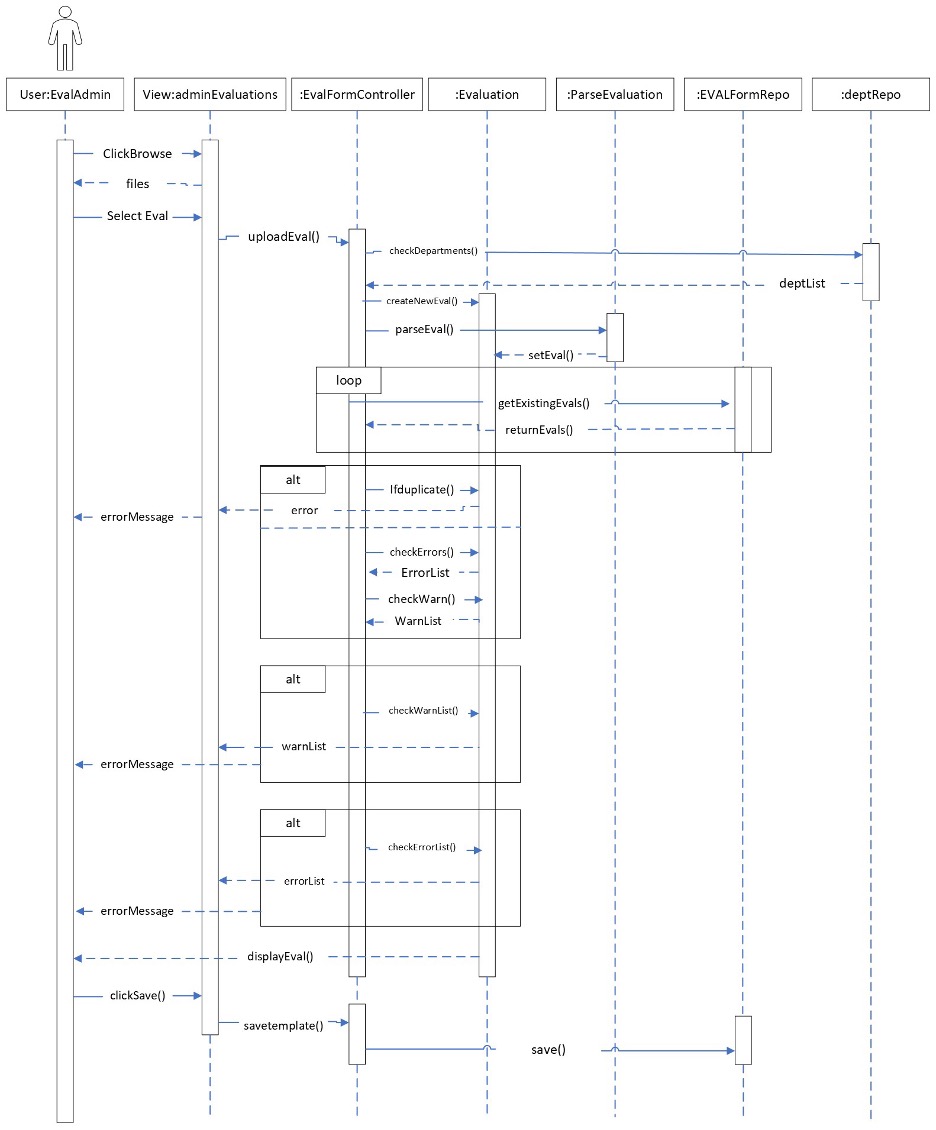
## Evaluator Edit Account Information



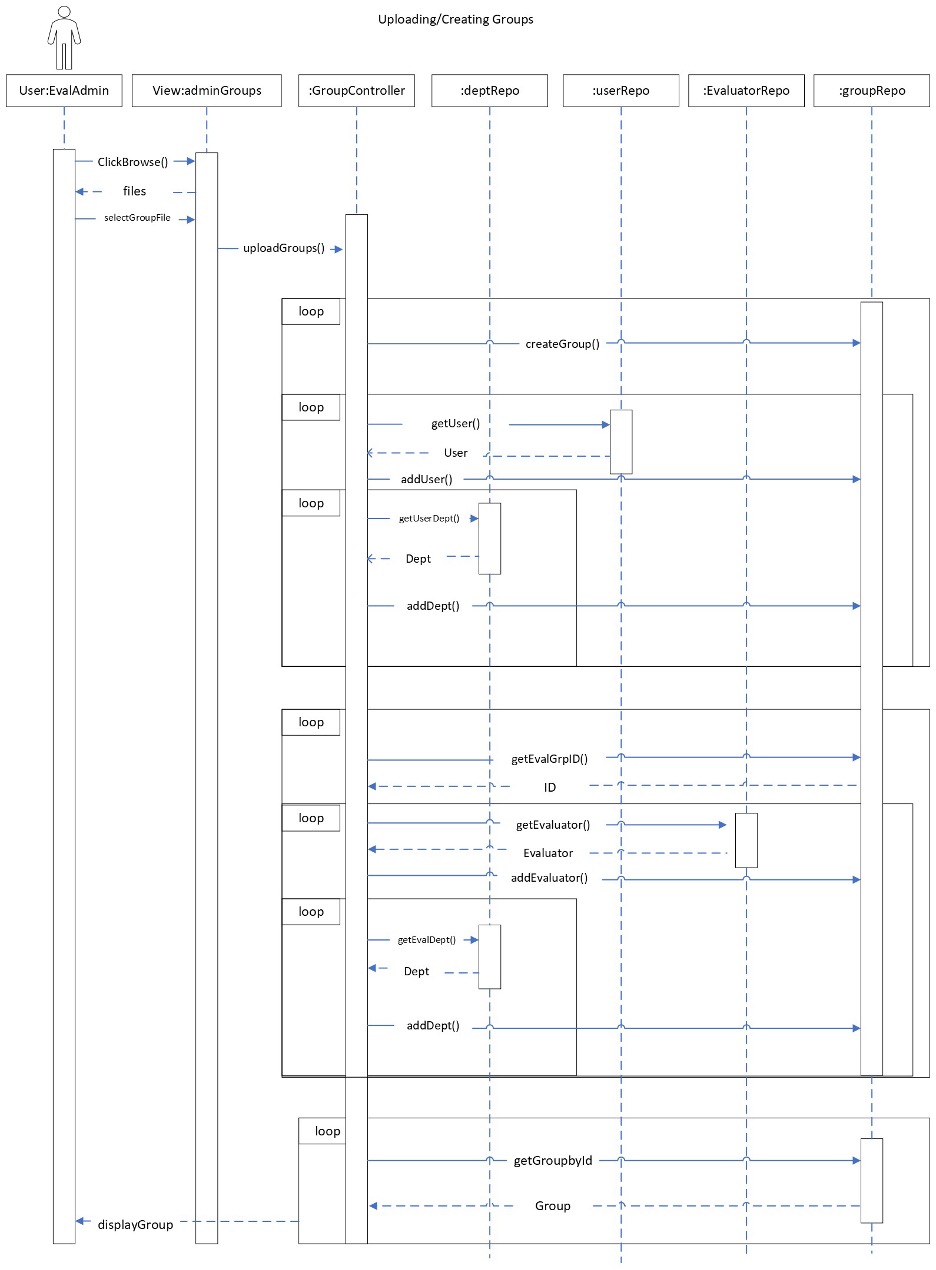
## Evaluator Perform Evaluation



## Uploading Evaluation Forms



## Uploading Groups



## Uploading Companies

