Design Document

Co-op Evaluation System

Senior Project 2014-2015

Team Members:
Tyler Geery
Maddison Hickson
Casey Klimkowsky
Emma Nelson

Faculty Coach:
Samuel Malachowsky

Project Sponsors:
Jim Bondi (OCSCE)
Kim Sowers (ITS)

Table of Contents

Table of Contents
Revision History
1 Introduction
1.1 Purpose
1.2 Scope
1.3 Overview
2 System Overview
2.1 Purpose
2.2 System Context
2.3 User Roles
2.4 Design Constraints and Limitations
3 Data Design
3.1 Data Description
3.1.1 Users, Departments, Colleges, and Configurations
Relational Model
Context
Element Catalog
3.1.2 Email Notifications and Logs
Relational Model
Context
Element Catalog
3.1.3 Evaluations, Forms, and Questions
Relational Model
Context
Element Catalog
4 Component Design
4.1 Reporting Service
4.2 Authentication Service
4.2.1 User
4.2.2 Privileges
4.2.3 EmployerAuthentication
4.2.4 Shibboleth
4.3 Form Service
4.3.1 FormCommand
4.3.2 AddForm
4.3.3 RemoveForm
4.3.4 QuestionCommand
4.3.5 AddQuestion
4.3.6 RemoveQuestion
4.3.7 UpdateQuestion

4.4 Email Service
4.4.1 Email Logging Service
4.4.2 Email Template Service
4.5 User Management Service
4.5.1 UserCommand
4.5.2 AddUser
4.5.3 RemoveUser
4.6 Evaluation Service
4.6.1 EvaluationCommand
4.6.2 AddEvaluation
4.6.3 UpdateEvaluationAnswers
4.6.4 UpdateEvaluationStatus
4.7 File Import Service
4.8 School Service
4.8.1 SchoolCommand
4.8.2 Add
4.8.3 Remove
4.9 Data Mapping
5 Human Interface Design
5.1 Overview of User Interface
5.2 Screen Images and Interactions
6 References
7 Appendices
Appendix A: Glossary
Appendix B: Issues List

Revision History

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Emma Nelson, Maddison Hickson, Casey Klimkowsky, Tyler Geery	Initial revision	December 13, 2014
v1.1	Casey Klimkowsky	Updated Element Catalog sections	February 17, 2015

1 Introduction

1.1 Purpose

The purpose of this document is to provide a detailed system design for the various components that comprise the new Co-op Evaluation System. While the Software Architecture Document provides a high-level structural view of the application and how the application interacts with external systems, this document focuses on just the system itself, and how its various software components are designed and how they interact with one another.

This document is intended to help the development team determine how the system will be structured at a detailed level. It is also intended for the project sponsors to sign off on the detailed structure before the team shifts into development. Finally, the project coach can use this document to validate that the development team is meeting the agreed-upon requirements during his evaluation of the team's efforts.

1.2 Scope

The current Co-op Evaluation System, an application used by OCSCE, has a number of performance, reliability, usability, and maintainability issues. Among others, session timeouts and submission timeouts are inherent problems of the current system. A new version started from scratch with up-to-date technologies needs to be developed.

The purpose of this project is to re-engineer the Co-op Evaluation System in order to leverage newer web technologies while also improving performance and user interaction. One of our primary goals is that by the conclusion of this project, we will, at a minimum, have supplied OCSCE and ITS with a product that is functionally equivalent to the existing system, but with fewer of the aforementioned issues. Time permitting, we hope to implement a small number of enhancements, as defined by our Software Requirements Specification.

1.3 Overview

This document provides a general description of the functionality, context, and design of the project, and addresses the system design from several viewpoints. The first design aspect is a comprehensive relational model, which outlines how information will be organized in the system's database. This is then followed by a detailed design of each of the system's components, which is achieved through UML diagrams, such as class diagrams and sequence diagrams, and accompanying descriptions. Finally, the last element of design addressed in this document is user interface design. Wireframes are provided as a concept of what the system's user interface will ultimately look like.

2 System Overview

2.1 Purpose

The purpose of the Co-op Evaluation System (CES) is to allow students to provide feedback on their most recent co-op, and for employers to provide feedback on a student's performance during their most recent co-op. Additionally, the system is used by faculty to approve or fail a student's co-op, and is also used by OCSCE to gather data on students' co-ops.

For details on the system's functionality, please refer to the Software Requirements Specification.

2.2 System Context

The below diagram shows the basic flow of data into and out of the system at a high level. Our system and direct interfaces are represented inside of the blue container, with the outside entities depicting how data is created and imported into our system. In this diagram, the "Co-op Evaluation Database" represents the relational database used to store system information, which is outlined in Section 3. The box labeled "Co-op Evaluation System" represents the core functionality of the system, which is broken down as a series of components in Section 4.

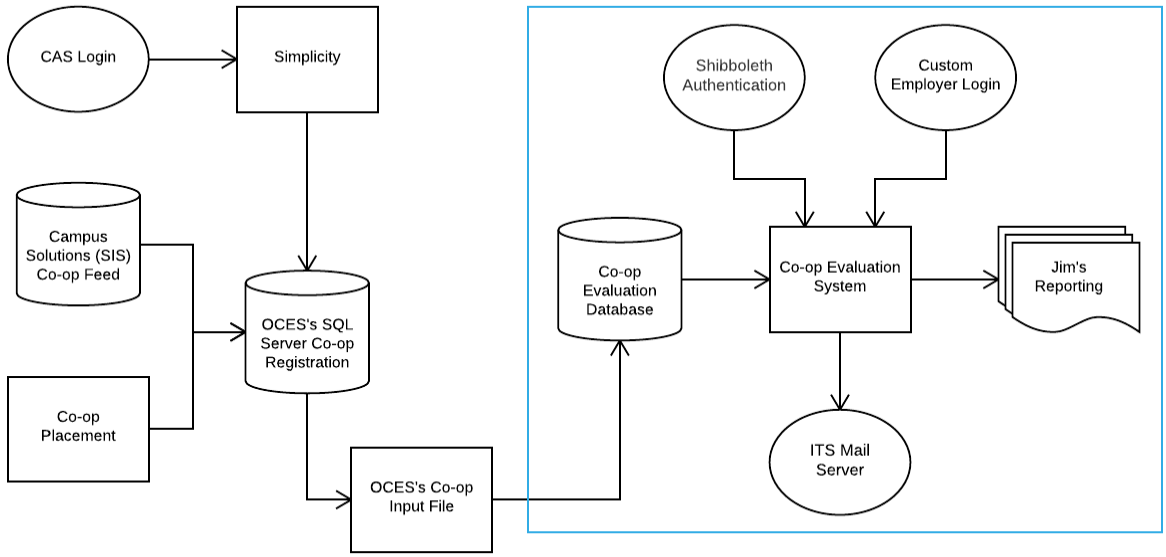


Figure 1. The flow of data into and out of the Co-op Evaluation System

For further information on the architectural design and its influences, please reference the Architecture Document.

2.3 User Roles

The following user classes represent the four main roles that users have when interacting with the system, which is done through the use of a desktop or mobile computer:

Student

A student uses the application to fill out a Work Report following a co-op block.

Employer

An employer uses the application to fill out an Employer Evaluation for a student following a co-op block.

Evaluator

An evaluator uses the application to review the student's and employer's evaluation submissions to determine the student's grade (S or F).

Administrator

An administrator uses the application to perform various administrative tasks, including gathering statistical data from evaluation submissions. Administrator has access rights to all departments and colleges.

For details on the functionality associated with each of the user roles, please refer to the Software Requirements Specification.

2.4 Design Constraints and Limitations

The system must comply with the development guidelines provided to us by ITS, as defined by the EWA Student Development Guidelines wiki page. At a high level, these guidelines include approved application frameworks, build tools, application server technologies, database standards, and several other technology standards.

3 Data Design

3.1 Data Description

3.1.1 Users, Departments, Colleges, and Configurations

Relational Model

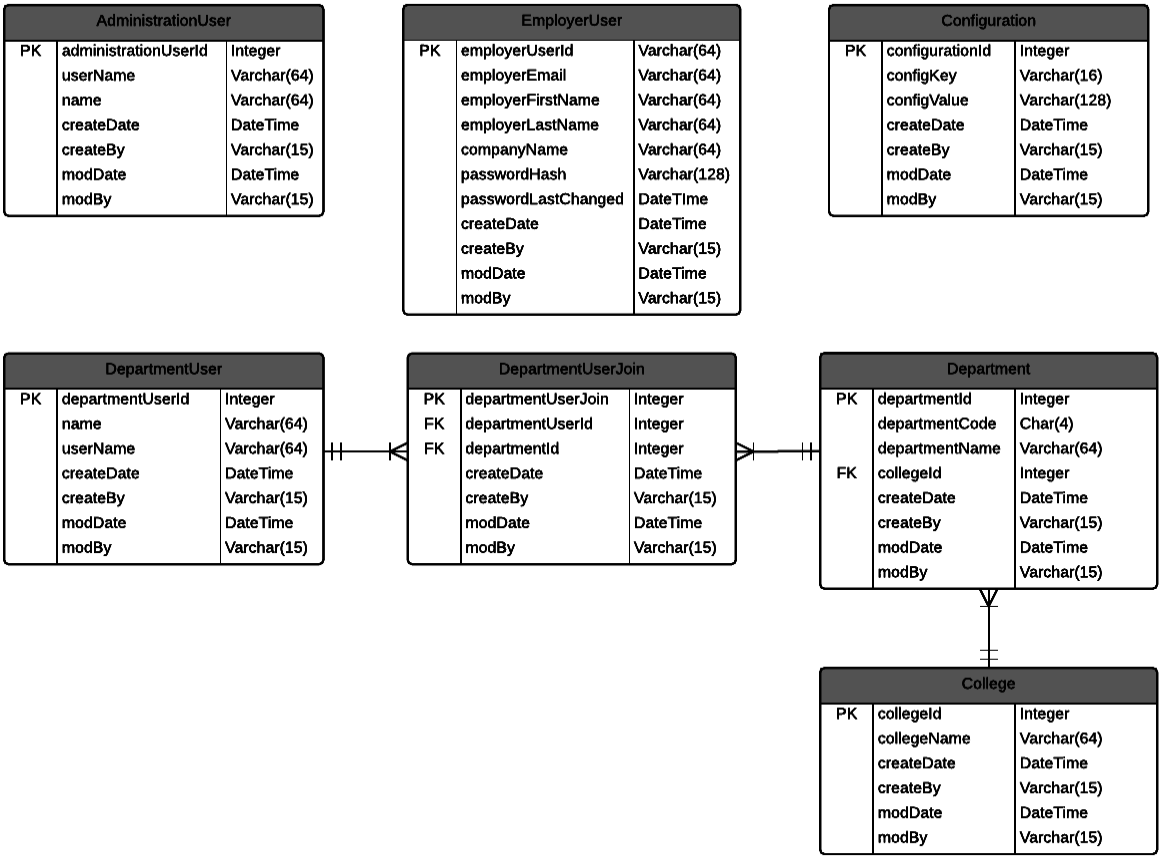


Figure 2. Relational model for users, departments, colleges, and configurations

Context

These tables are concerned with functionality around system users, departments and colleges, and miscellaneous configurations needed by the system.

Element Catalog

AdministrationUser

Contains administrative users, including the administrator's University ID and full name.

EmployerUser

Contains employer account information. Since employers are not authenticated through Shibboleth, their account information has to be stored in this system. An employer account is

comprised of their email address, and a password, which is stored as a hash. The date when the employer last changed his or her email is stored here as well, in case there is a need to have employers periodically change their password.

DepartmentUser

Contains department users, including the user's University ID and full name.

DepartmentUserJoin

Associates departments with department users. A user may be a part of multiple departments, and a department has many users, which is why there was a need for a join table.

Department

Contains departments at RIT, including the department name and code, and the name of the college the department belongs to.

College

Contains colleges at RIT, including the unique college acronym (e.g. GCCIS).

Configuration

Contains any configuration values that are required by the system, and must be saved within the database itself.

3.1.2 Email Notifications and Logs

Relational Model

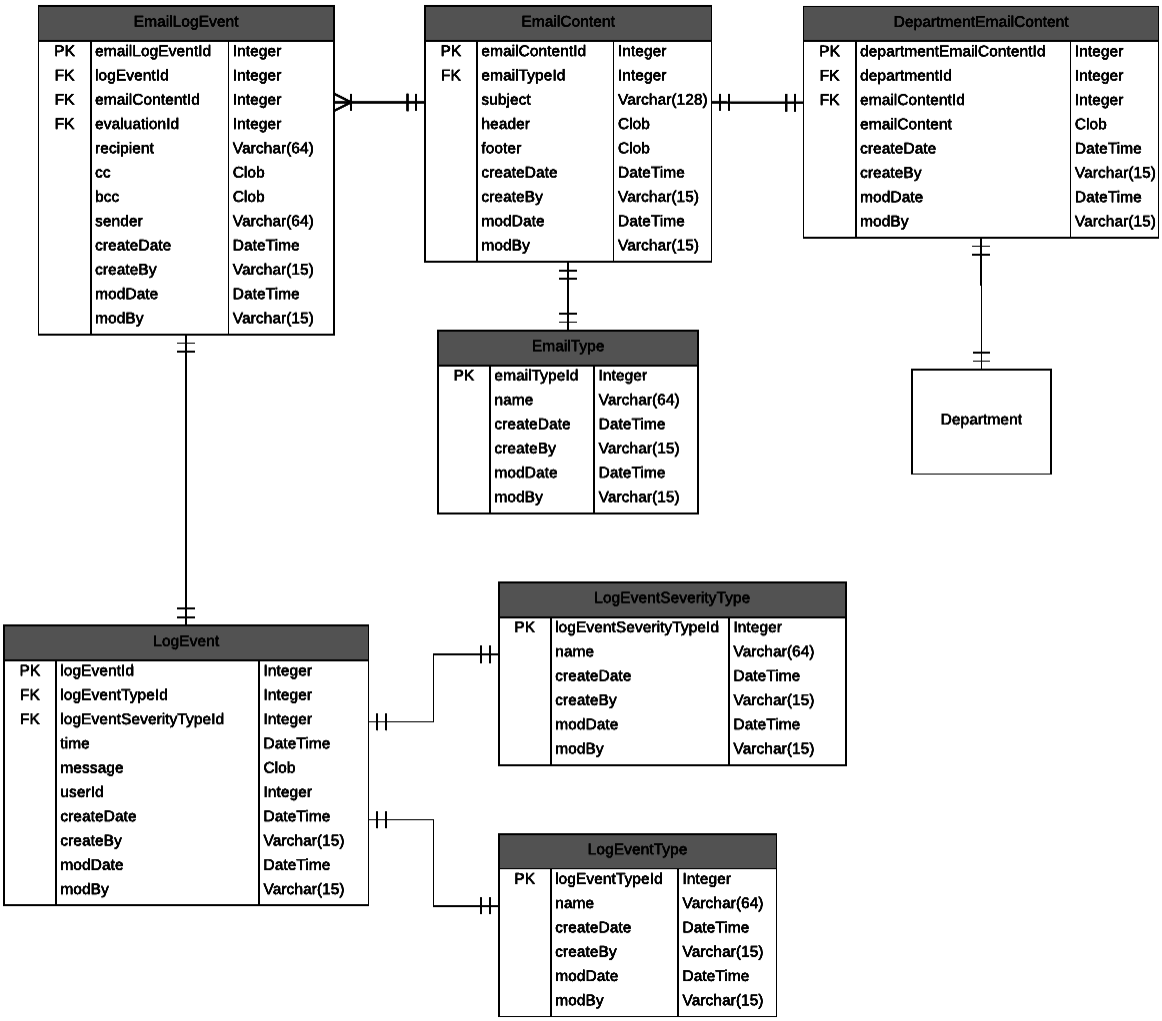


Figure 3. Relational model for email notifications and logs

Context

These tables are concerned with functionality surrounding email notifications, and various events within the system.

Element Catalog

LogEvent

A log entry representing an event in the system including a date and a time, a message explaining the event, which user caused the event, the type of event, and the severity of the event.

LogEventSeverityType

This table contains a list of all of the different severity levels a log entry can be.

LogEventType
This table contains a list of all of the different event types that can occur in the system.

EmailLogEvent
Contains information about an email, including the subject line, sender, receivers, and a link to the content.

EmailContent
The content of an email, including the subject line, header, footer, and the time it was last updated.

EmailType
The type of an email (e.g. Reminder or Employer Login).

DepartmentEmailContent
Contains special department-specific information in an email.

3.1.3 Evaluations, Forms, and Questions

Relational Model

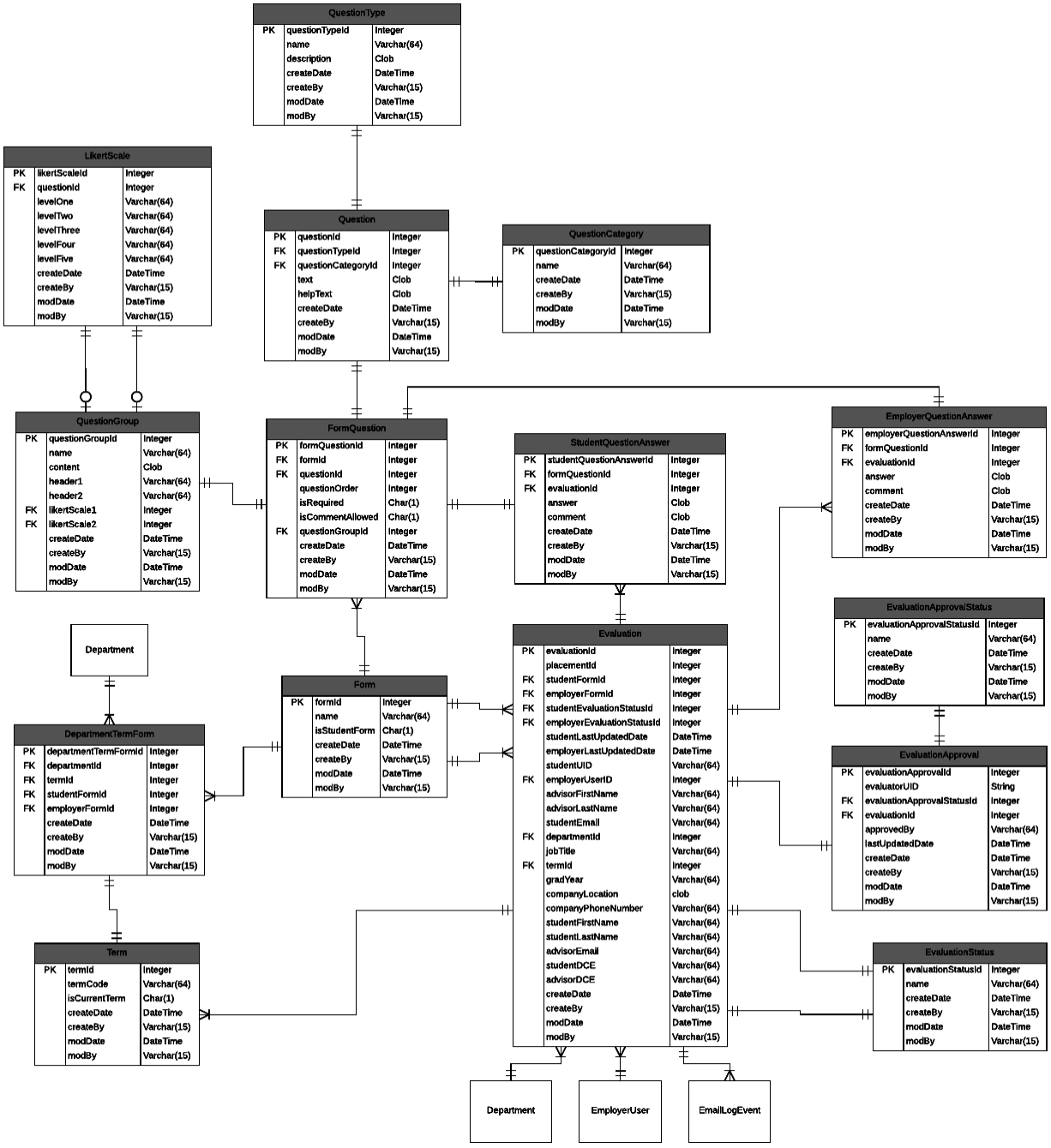


Figure 4. Relational model for evaluations, forms, and questions

Context

These tables are concerned with functionality surrounding forms, form questions, and form answers. Forms are associated to departments using the DepartmentTermForm, which is

related to the Department table in Figure 2. This part of the data design is also concerned with evaluations, and the status of an evaluation. Evaluations are associated to the employer that completed an evaluation through the EmployerUser table in Figure 2.

Element Catalog

LikertScale

Contains Likert scales that are associated with the Likert and Double Likert questions in a question group. This table is required, as different Likert questions may have different scales associated with them (e.g. 5=Extensive, 3=Moderate, 1=Minimal and 5=Excellent, 3=Average, 1=Poor). Each Likert Scale question group will have one scale associated with it, and each Double Likert question group will have up to two scales associated with it.

Question

Contains questions that are asked on forms, including the question text, and a reference to which category the question falls under. The questionType field is a reference to a Java enum that will contain all the possible question types (e.g. Likert).

QuestionType

An enum that represents all of the possible question types (e.g. Numeric).

QuestionCategory

Contains the question categories that can be asked on forms, including the name of the category (e.g. Ethics).

QuestionGroup

A QuestionGroup associates a number of questions with each other. Each QuestionGroup has a short name, and a content field (which is what is actually displayed on a form). The header1 and header2 fields are optional, as they are only used with Likert and DoubleLikert questions.

FormQuestion

Contains the questions to be asked on a form. Each FormQuestion has a reference to a Question via questionId, and a reference to the Form it belongs to via formId. Additionally, each FormQuestion has a questionOrder field which contains an integer which specifies what the ordering of the questions, and the overarching category that the FormQuestion is connected to via categoryId. Each FormQuestion also has a questionGroup, which is used to group a set of questions on a form; this field contains the text displayed above a group of questions.

StudentQuestionAnswer

Contains student answers to questions on forms, including a reference to the Question and FormQuestion it answers, and the Evaluation it is an answer for.

EmployerQuestionAnswer

Contains employer answers to questions on forms, including a reference to the Question and FormQuestion it answers, and the Evaluation it is an answer for.

Form

Contains forms to be completed by students or employers, including the college name.

DepartmentTermForm

Associates specific forms to the department they are used by and the term are being used for.

Term

Contains RIT semester terms, including the term code (e.g. 2141).

Evaluation

Contains references to the student form, employer form, student form status, employer form status, and the times the student and employer last updated the evaluation. Also contains an evaluation ID, placement ID, student UID, employer contact ID, and other related information. An Evaluation is associated with one or many EmailLogEvent objects, as these keep track of when notification, confirmation, and other related emails are sent out to the associated student and employer.

EvaluationStatus

Contains possible evaluation statuses, including the status name (e.g. Saved).

EvaluationApproval

Contains references to the evaluations and connects that evaluation to a specific evaluator. This table also connects that evaluation to a specific evaluation approval status.

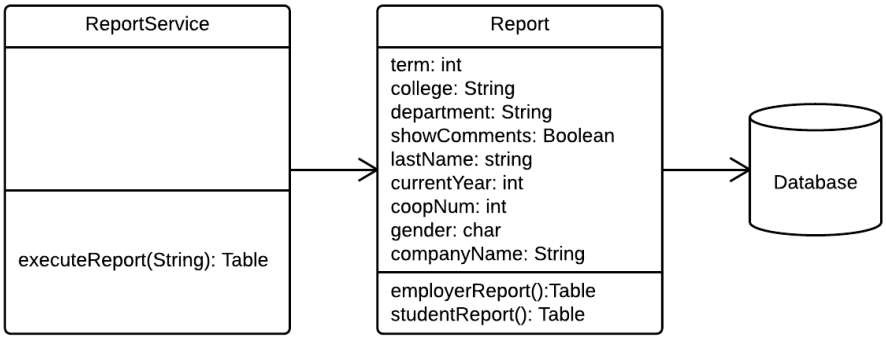
EvaluationApprovalStatus

Contains possible evaluation approval statuses, including the status name.

4 Component Design

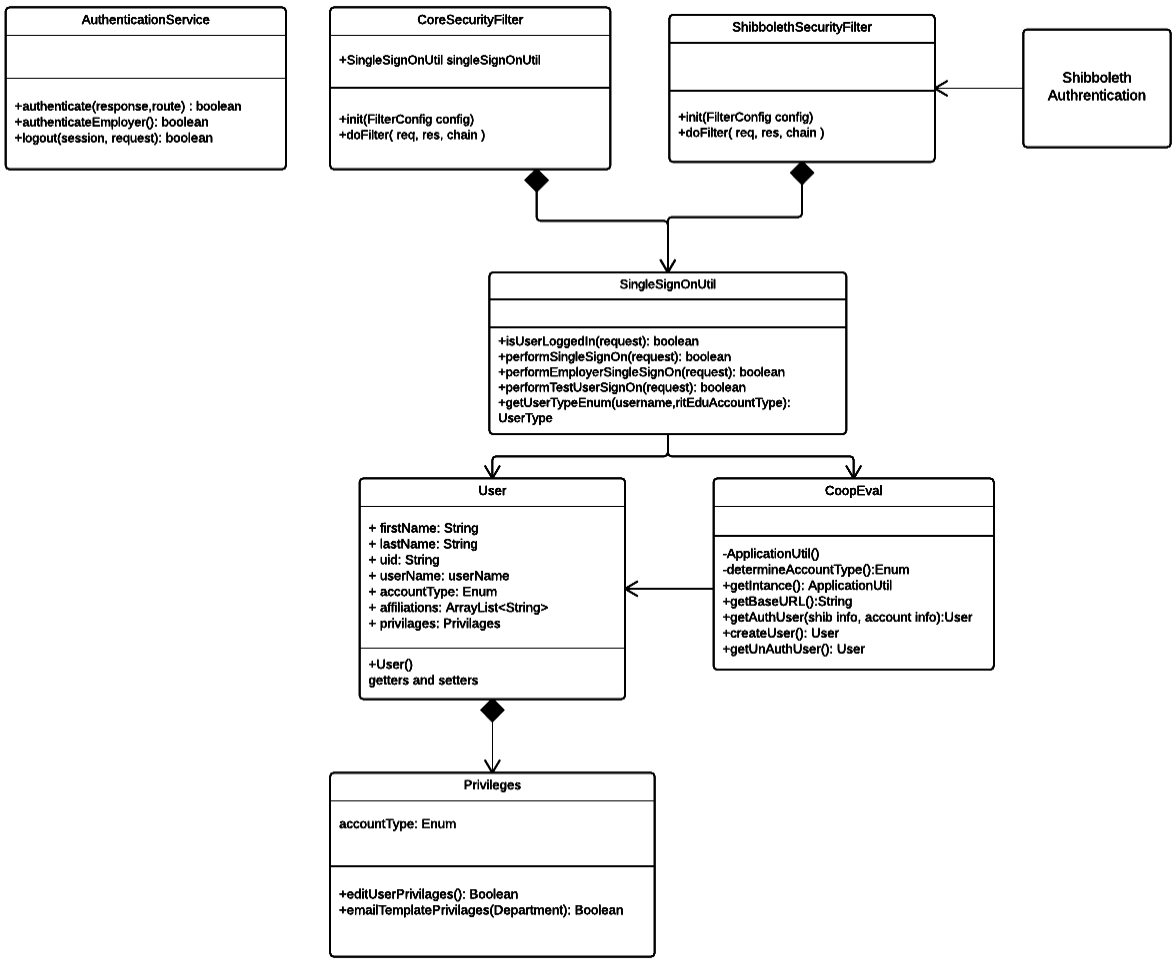
4.1 Reporting Service

ITS will be able to create a view of the system's database, which can be used by an external reporting tool to generate dynamic reports. Our system will use Active Query Builder to create simple table reports that can be viewed within the system.



4.2 Authentication Service

Using Shibboleth and the system's EmployerUser table, this service will identify if a user is within the system and, if so, log them in. At the same time, it will also identify what the user's privileges are and act accordingly (e.g. if the user is a student, administrative content will not be displayed).



4.2.1 User

Our system will take information given to us by Shibboleth or our employer authentication system and create a specific User object that can be accessed by the entire system. The system will use this information to specify an enumeration to that user, which will define what that user's privileges are. This User object is stored in the session.

4.2.2 Privileges

As stated above, our system will use an enum to associate each user type to each individual user. This enum will define the user type; for example, "1" could represent an employer, "2" could represent an administrator of the system, and so on. The User object contains a Privileges class, which returns true or throw an unauthorized error for user-specific privileges that are dependent on the user type (or enum).

4.2.3 AuthenticationService

The AuthenticationService contains endpoints that perform login for shib or employers. The login methods do nothing themselves but are a way for the Filters to check for the specific login URI's and perform the login functionality.

4.2.4 CoreSecurityFilter

The CoreSecurityFilter is hit on every request to any html or controller endpoint. If the request is attempting to hit AuthenticationService's employer login endpoint the filter will call SingleSignOnUtils method for logging in employer users. It also makes sure the request isn't whitelisted. For any other request the session is checked to see if the a the user has been authenticated. If the user has been authenticated then the filter lets the request go through, if the user is not authenticated an unauthorized error is send back.

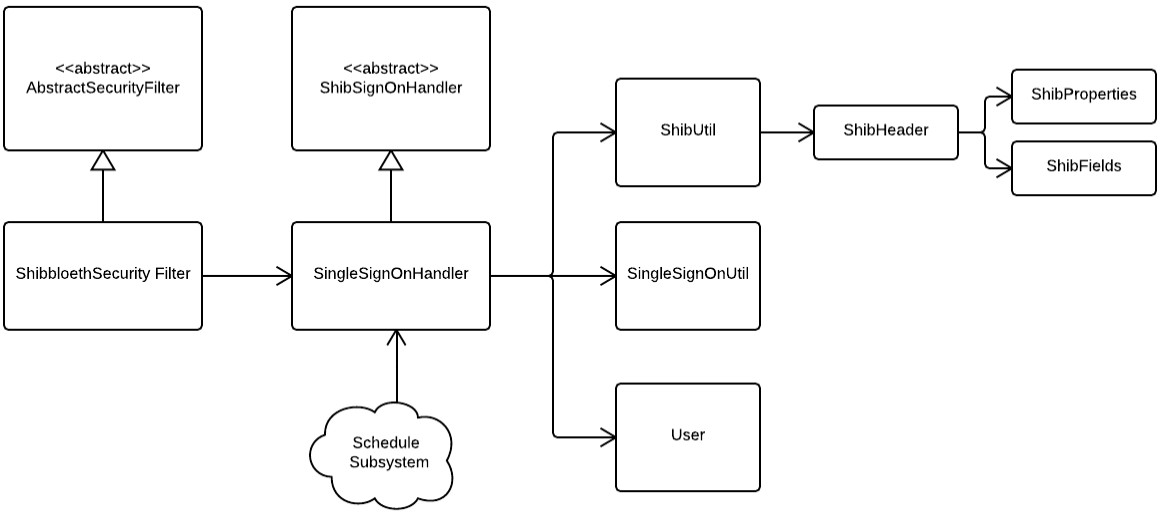
4.2.5 ShibbolethSecurityFilter

The ShibbolethSecurityFilter is only called when the shibboleth login endpoint is called in AuthenticationService. This filter takes in the shibboleth header information and verifies that the User has access to the system.

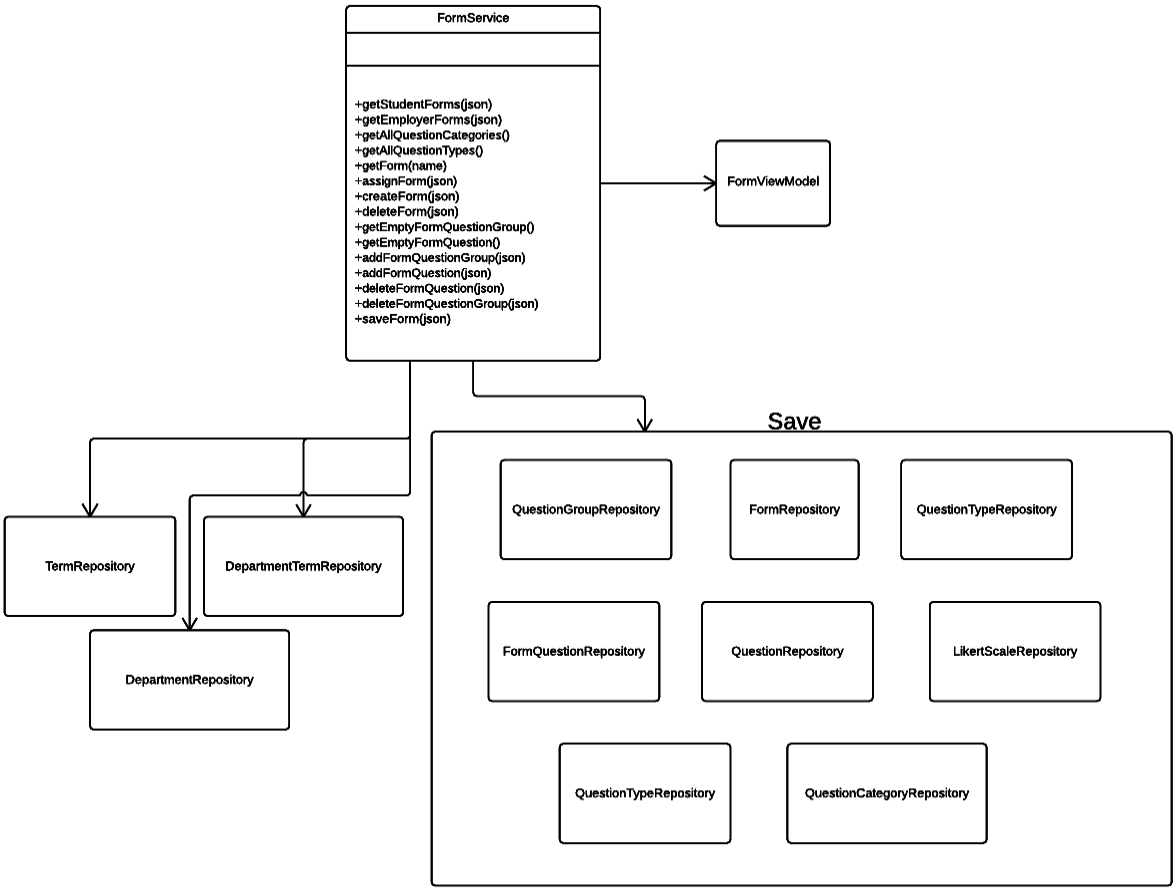
4.2.5 SingleSignOnUtil

This class is in charge of either taken in the Shib header information and turning it into user that is stored in session or taking the login credentials of an Employer and authenticating the user. The SingleSignOnUtil uses the department and admin user tables to verify whether a shib user has access to the system and also uses the employer table to check if the employer's login credentials are correct. The SingleSignOnUtil can also check if someone is logged into the system by checking if a user is stored in session. SingleSignOnUtil is also in charge of figuring out which UserType enum to assign a specific user.

4.2.6 Shibboleth



4.3 Form Service



4.3.1 FormService

This controller is in charge of taking in requests for actions related to FormService. The two major components of FormService are editing forms and assigning forms to a department. For form editing the approach that was taken that instead of persisting every time a change was made the methods actually alter the formViewModel that is stored in the session. Once the save is called the formViewModel is compared with the form in the database in order to use the repositories to persist the necessary changes.

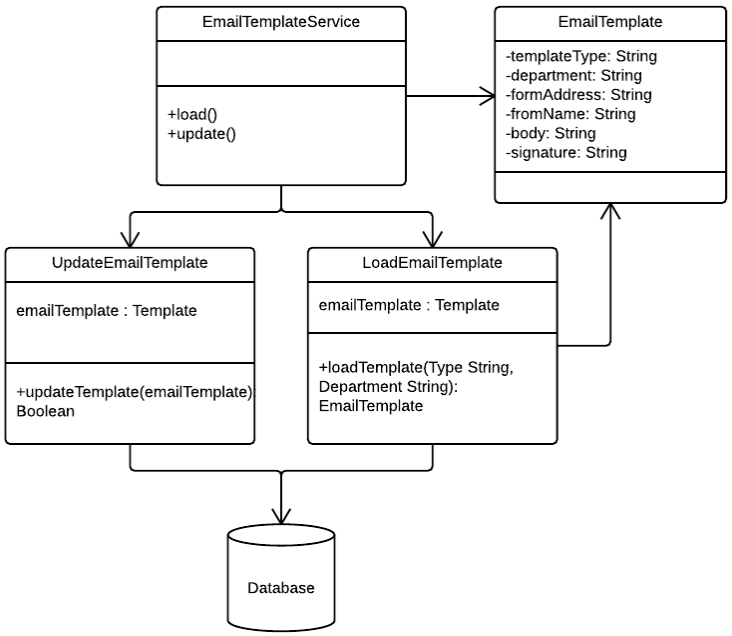
4.4 Email Service

The Email Service will be in charge of sending emails and editing email templates. These emails will be sent out automatically, but can be manually overridden.

4.4.1 Email Logging Service

The Email Logging Service will be in charge of logging events within the system's database. Events, such as emails being sent out, will be logged. These logged events will be kept within a text file.

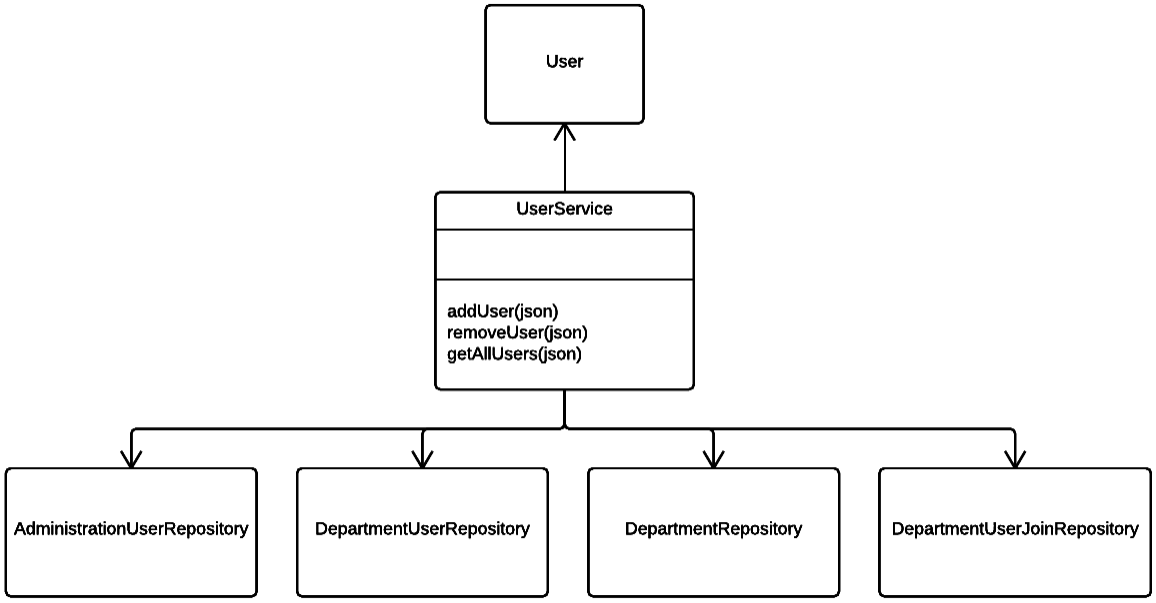
4.4.2 Email Template Service



The Email Template Service is the controller for all actions associated with email templates. The Email Template Service is in charge of updating the view and delegating commands from the view. The Email Template Service will be called by the view directly. The EmailTemplate class is used to contain all the information related to an email template row in the database. UpdateEmailTemplate is the class responsible for updating a specified template within the database. Lastly, LoadEmailTemplate will find a specific email template from the database and send its associated information to the Email Template Service.

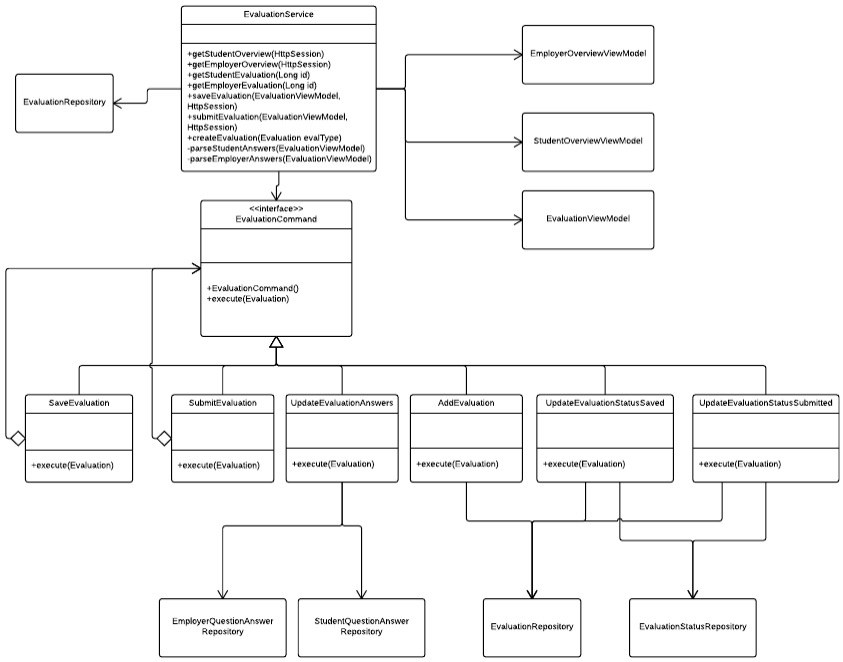
4.5 User Management Service

The User Management Service is in charge of giving or removing administrative and department access to specific users. This will be done by storing the user's UID into the admin or department table within the system. The User Management Service will be responsible for creating the User object with the information from the client.



4.6 Evaluation Service

The Evaluation Service will be responsible for handling all evaluations that are being tracked by the system. An evaluation is either completed or approved. For this service, we will be making use of the command pattern.



4.6.1 EvaluationCommand

This class contains an execute method, which handles an Evaluation object and passes it to a designated class that knows what to do with it.

4.6.2 AddEvaluation

A “receiver” class, which handles adding an Evaluation to the system by persisting it to the database.

4.6.3 UpdateEvaluationAnswers

A “receiver” class, which handles updating question answers in the system by persisting it to the database.

4.6.4 UpdateEvaluationStatusSaved

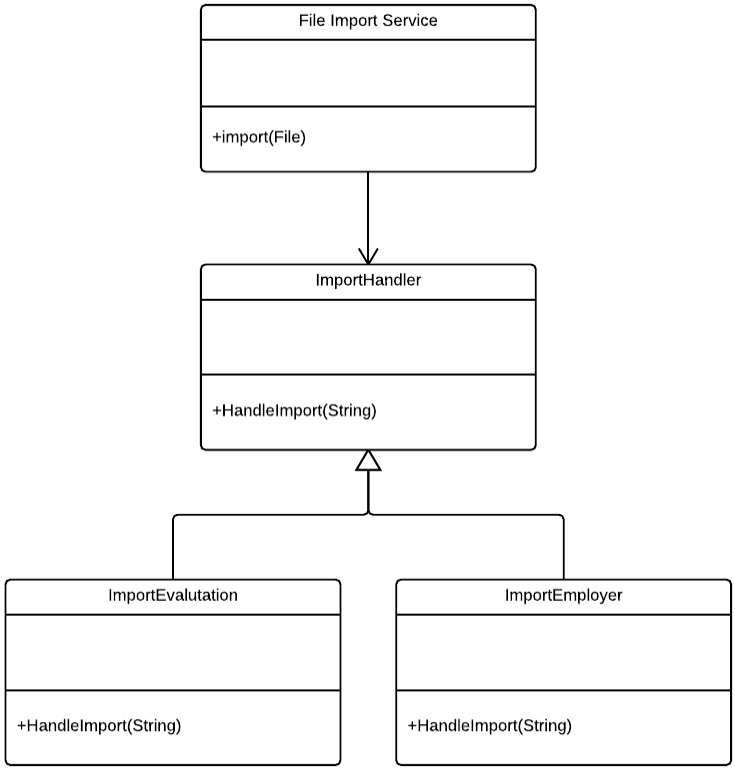
A “receiver” class, which handles updating an Evaluation object’s status to saved in the system by persisting the changes to the database.

4.6.5 UpdateEvaluationStatusSubmitted

A “receiver” class, which handles updating an Evaluation object’s status to submitted in the system by persisting the changes to the database.

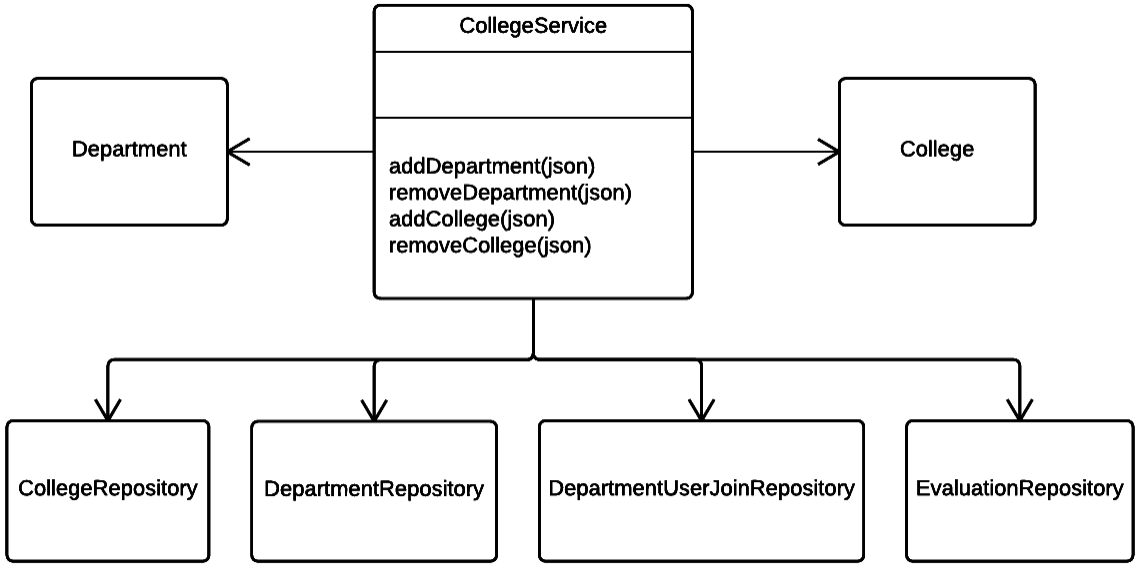
4.7 File Import Service

This service handles importing the file given to us from Jim’s system, and will be used to populate data into our system. If a new user has registered for a co-op and is passed into our system, this service will handle creating an Evaluation object and associating it with that user.



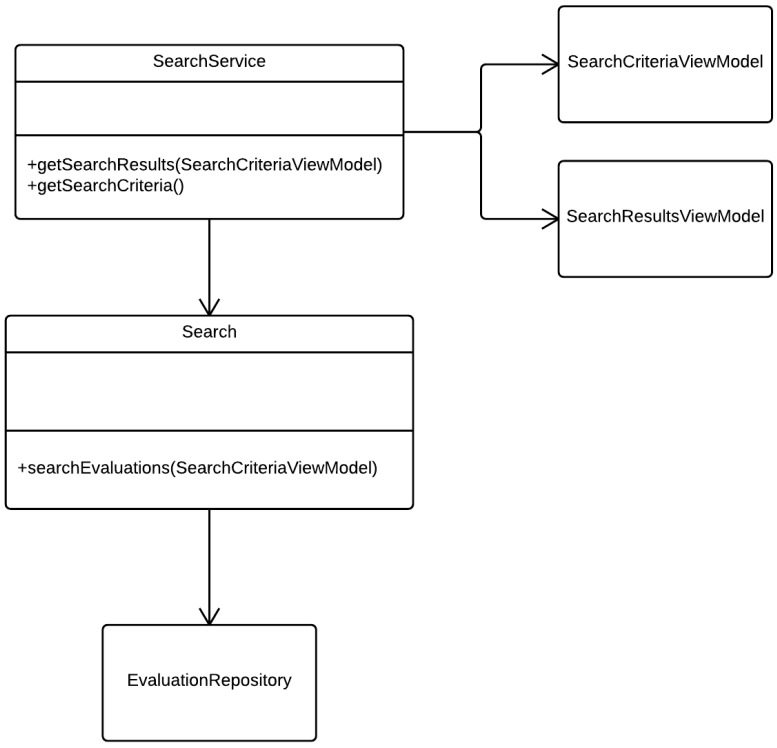
4.8 College Service

The College Service handles all logic dealing with Colleges and Departments. This logic includes deletion, addition, and simple retrieving of colleges and departments. The data is retrieved from the database using the repositories shown in the diagram below.



4.9 Search Service

The Search Service will be responsible for handling all evaluation searches.



4.9.1 Search

This class takes in a SearchCriteriaViewModel and parses that viewmodel into the necessary values to query the database. This class then will return all the Evaluations that match the criteria results.

4.10 Repositories

Repositories are used to access data from the database. The repository's name will indicate which table it is accessing.

5 Human Interface Design

5.1 Overview of User Interface

One of the major pain points of the current system is its poor usability, and we aimed to remedy that while designing the user interface of the new system. Although we referenced the RIT Web Standards document for colors, fonts, and other guidelines we are expected to work within, we used this document minimally since it was last updated in 2011. In order to design to fit where RIT's web styles are currently headed, we used newer websites within the RIT domain or websites that had recently been updated as our primary inspiration for the new

user interface. The OCSCE website was the biggest influence, as it has recently been updated, and the Co-op Evaluation System is a part of their suite of applications. We wanted to make sure it fit in and used similar navigation formats. Other websites used as inspiration include PawPrints, a new website built by RIT Student Government, and the website for the RIT Honors Program, which has been given a new user interface within the last year.

5.2 Screen Images and Interactions

All our wireframes were created in Lucidchart and are available on our website or here.

We chose not to include them directly in this document in order to keep it from becoming too long, and to minimize loading time in the live Google Drive version.

6 References

[1] A. Shvets, G. Frey, and M. Pavlova. *SourceMaking*. [Online]. Available: http://sourcemaking.com

[2] E. Gamma , R. Heml, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Indianapolis, Indiana: Addison-Wesley Professional, 2002.

[3] *RIT Web Standards*, Rochester Institute of Technology, Rochester, NY, 2011.

7 Appendices

Appendix A: Glossary

Term	Definition
ITS	Information and Technology Services
OCSCE	Office of Career Services and Cooperative Education
RIT	Rochester Institute of Technology

Appendix B: Issues List

The team is using Trello to track issues; however, below you will find a high-level list of outstanding issues with this document. If finer detail is required, please reference the team Trello board, activity tracker, and/or Google Drive.

Number	Priority	Description