Lane Community College Co-op Listing Website

Technical Design Document

*Version 0.5*

Introduction

**Purpose**

The purpose of this document is to provide an overview of the design of a website to allow for development to begin with the understanding of what it is to be developed and how it will be developed. This document provides information needed to get a description of the website that is to be developed.

This document will be updated and refined as requirements change and ideas become clearer.

**Scope**

This document will cover the most basic design elements of the website and its components. This may include a database schema, class diagrams, a functionality overview, and technologies used along with the website. It will not cover how to install or set-up required technologies, deployment of the website, or any information pertaining to upkeep of the website.

**Audience**

The intended audience for this document are Lane Community College Co-op Coordinators, students, the project development team, and website maintainers.

Design Overview

**Problem Domain Description**

Lane Community College students are often required to go through some form of co-operative education for their degree. There is no way for students to find a co-op opportunity short of their co-op advisor sending e-mails. This can make finding a co-op opportunity fairly challenging.

**Functionality**

To help with solving the outlined problem, the website will have the following minimum functionality requirements:

* List Co-op opportunities and their full details, preferably grouped by the degree best suited for the opportunity.
* Allow co-op coordinators to freely add, remove, and edit the opportunities associated with the majors they advise.
* Allow students to register for the site by having a co-op coordinator send a registration invitation to their e-mail.
* Allowing students to apply for a co-op directly through the website by attaching their resume and any other files that may be required to apply.
* Allowing businesses to apply for accounts so that they may directly submit co-op opportunities through the site, to be approved by the site administrator.

These requirements alone will solve the biggest part of the problem: having no centralized or organized listing of co-op opportunities. They are not the only functionality the site is expected to have, though. Further functionality that would be preferred includes:

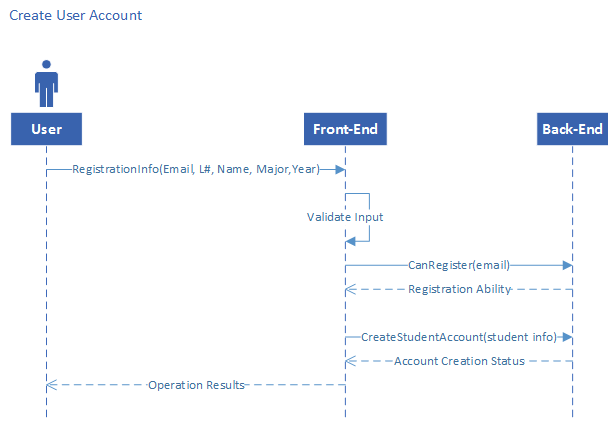
* Allowing students to check off all courses they have completed
* Displaying only co-op opportunities to students who meet the minimum courses required to have the skills needed for the opportunity.
* Displaying co-op opportunities to students based on their selected major

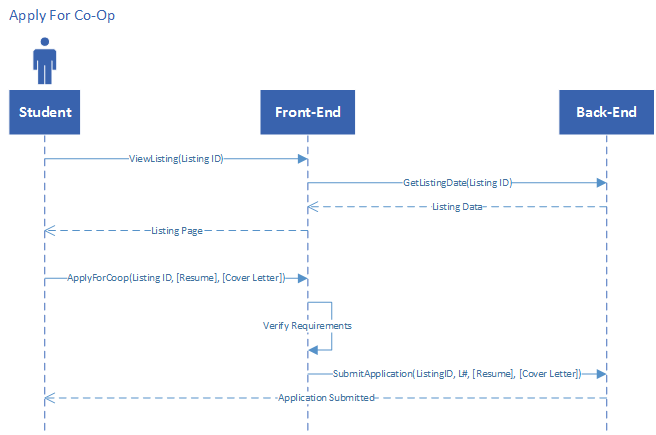
More functionality may be added or altered as discussion continues.

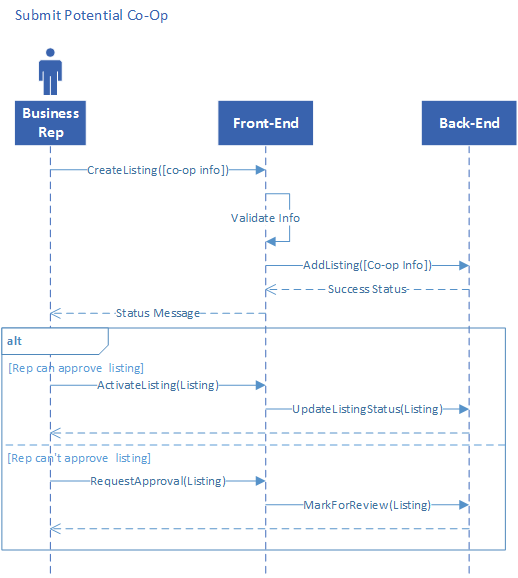
**Use Case Diagram**



**Sequence Diagrams**

****

****

****

**Approach**

The approach to designing this website will take place over multiple phases:

* Planning Phase – The planning phase involves brainstorming ideas and requirements for the website to be feasible for its intended purpose.
* System Design Phase – The system design phase is where a base prototype of the project is created, intended to change over time. This serves as a basis for future development.
* Testing Phase – This phase, as the name implies, will be all about testing the website. Unit testing with a dummy database will be done to ensure all site functions work as intended, and nothing can be broken short of forcing the site down.
* Implementation Phase – Implementation will include getting the website up and running in a usable, if rough, state. Modifications may still be made to the website in this phase.
* Finalization Phase – The end of the base website creation, this phase will involve deployment for proper use. While called the finalization phase, tweaks can still be made to the website at this time. This phase simply means that the original functions the website was created for are working without issue, and that it can be used in full.

These phases do not necessarily happen in order, and can be revisited at any time, and/or take place in parallel.

**Outstanding Issues**

This section will layout current issues that are unresolved at the time this documented was created. One of the issues that still remains is whether or not the site will be used with myLane for student verification and authorization. The other issue is getting the site approved to be hosted on the current Lane Community College website. These issues can vastly change the implementation of the website if they are or are not resolved.

UPDATE 2016/28/03: No confirmation on the site being hosted by Lane Community College has been given, so a major change has been made: switching to use ASP.NET instead of PHP.

Design methods and Standards

**Programming Standards**

The language the site will be created in is ASP.NET/C#. HTML and CSS will be used for design aspects, and JavaScript will be used for client-side functionality. These guidelines will be followed while using C# or JavaScript:

* Commenting - All functions will include comments that briefly describe what the function does or is used for.
* Formatting – All code will have proper indention for easy reading and cleanness of code. Indents will be set to 4 space characters, and not tab characters. Curly brackets will go on a new line after the function declaration.
* Casing – Functions, Class Names, and Properties will use Pascal Case (e.g. MyAwesomeFunction). Variables will use Camel Case (e.g. bestVariable).

**Software Development Tools**

During the development time of the site, the tools that will be used will consist of the following:

* Visio – While designing and building the website, illustrations will often be used to get a preliminary idea of what to do. Visio will be used to create these illustrations.
* Visual Studio – C#, HTML, CSS, and JavaScript can be written using Visual Studio. Visual Studio provides text edition, auto-completion, syntax highlighting, and a great debugger.
* git – To easily keep track of changes and access website code, the project will be using git for its version control.
* Visual Studio Team Services- To setup iterations and divide work among team members.

User Experience

**Organization**

When it comes to the User Interface for a company looking for an intern, they will first need to apply for an account. Once they have been approved for and registered an account, they will be able to upload co-op opportunities to the site. They will be able to select the department and/or majors that best fits their listing. They will also be able to add a GPA requirement, description of the position, and how to apply for the position.

Once a company has uploaded all the information for a co-op posting, it will then be up to the co-op coordinator to review the posting and approve it to be listed on the site. If the posting does get listed onto the site, the client will be able to update it if they hire someone for the position so that other students will know that the position has been filled.

**Co-op Coordinator**

The co-op coordinator will be able to upload job postings for companies that have not yet registered for the site, or decided not to register at all. They will be able to view potential job postings that are associated with the degree programs they advise. They will also be able to approve or deny co-op postings submitted by clients. If approved the posting will be visible to students. If the position has been filled, the co-op coordinator will be able to hide the job posting so that other students don’t apply for it.

Co-op coordinators will have the ability to send out registration invites for students and clients. Along with this, they will also be able to block or remove accounts that are no longer used on the site (i.e. graduated student).

**Student**

Students will be given access to the site from the co-op coordinator that advises their degree. A student will be able to upload their resume and/or any other required documents that apply to co-op opportunities. They will also be able to view all the co-op opportunities that are currently available for their major. The student will be able to apply for co-op opportunities directly from the site.

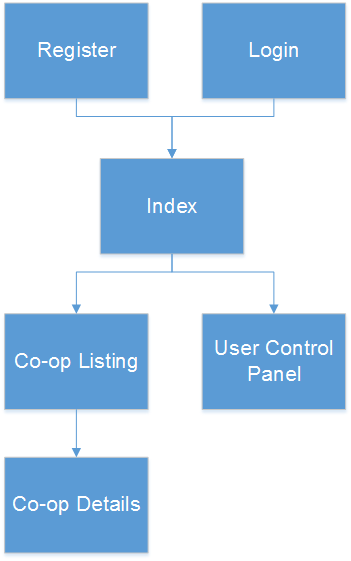
Website Structure

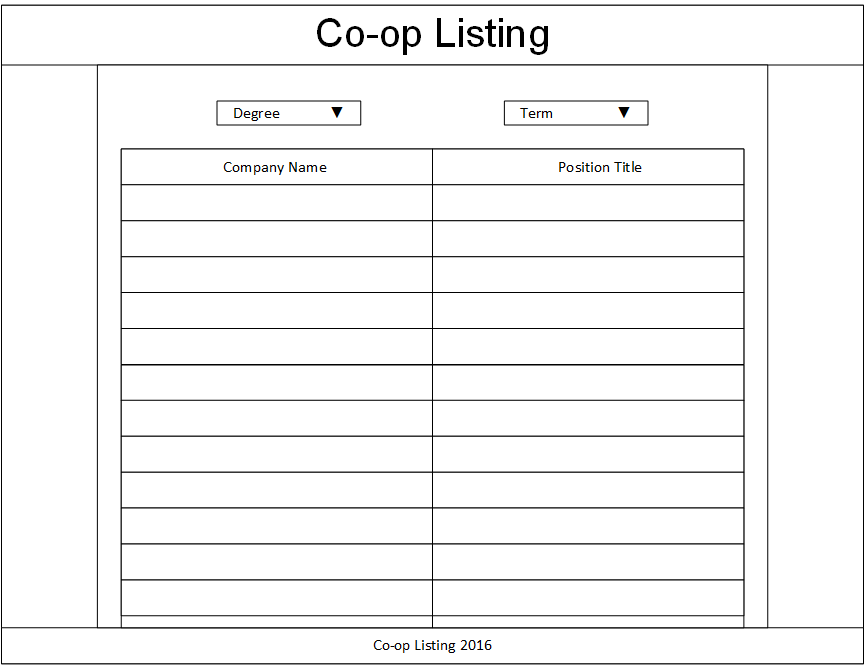
**Website Pages**

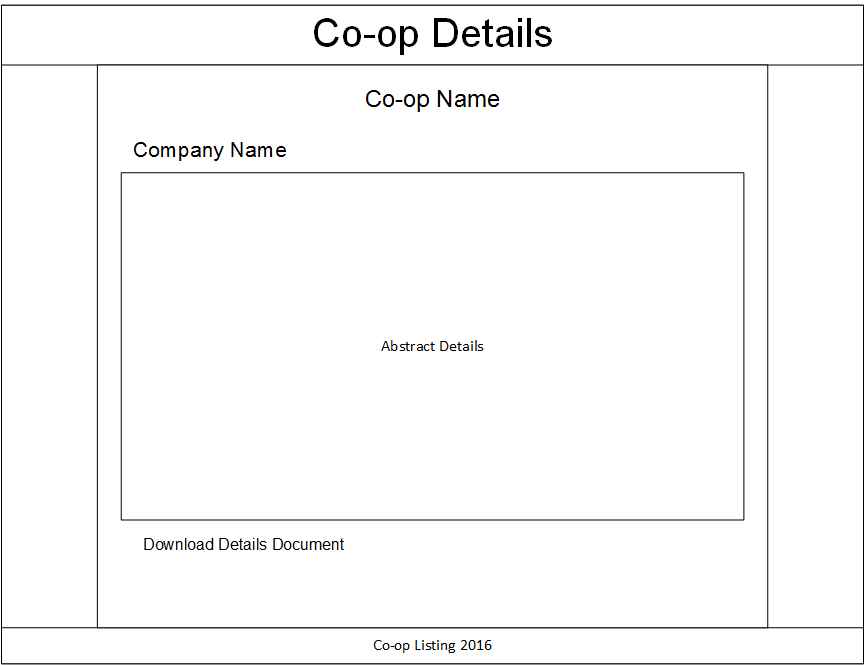
The basis of any website, the web pages are what the user sees and interacts with. For this website, there will be a minimum of six primary pages, and any number of possible subpages not currently thought out. These six pages are:

* Registration Page – This will be the first page almost all user who access the site will see for the first time. It will allow those that have been given a registration invite to set up their account.
* Login Page – Every time a current user accesses the website, they must first log in. That is what this page is for. Logins will be session based, so if the user closes their browser at any point, they will see this page the next time they access the website.
* Index Page – Normally the index page of a website serves as the base for information about the website. In this case, however, it will vary depending on the user account type accessing it. For students, it will default to the Co-op Listing page. For clients and co-op advisors, this will be their User Control Panel page.
* Co-op Listing Page – The whole reason this website is being created is for easy access to current co-op opportunities, and that is exactly what this web page will show. Students will be allowed to see co-op opportunities that relate to their major. Once selected, a table will be generated with a list of all currently open co-op opportunities for the Students selected degree, linking to a details page.
* Co-op Details Page – To avoid too much clutter on a single page, the larger details for co-ops will be viewable on a separate page. This page will include the description, requirements, and any files associated with the co-op listing.
* User Control Panel Page – The User Control Panel will be unique to each account type. Students will be able to change their account information here, including uploading their resumes and setting completed courses. Clients will also be able to edit their information from here, with the addition of submitting new co-op opportunities. As for co-op coordinators, this is the page that will let them manage all aspects of the department they advise. From accepting or declining submitted co-ops, to sending out registration invites, this page will be their main hub.

**Site Map**

****





Data Design

Inheritance Strategy & Object Relational Mapping (ORM) between entities

(All considerations listed)

The following tables showcase; description, additional notes (if any), pros and cons of each strategy under consideration.

|  |  |
| --- | --- |
| Single Table Mapping | |
| DESC: | The Inheritance strategy; “Single Table” maps all classes in the hierarchy to the base class' table. |
| NOTE: | Mapping subclass state to the superclass table is often called *flat* inheritance mapping. |
| PROS: | Single table inheritance mapping is the fastest of all inheritance models, since it never requires a join to retrieve a persistent instance from the database. Similarly, persisting or updating a persistent instance requires only a single INSERT or UPDATE statement. Finally, relations to any class within a single table inheritance hierarchy are just as efficient as relations to a base class. |
| CONS: | The larger the inheritance model gets, the "wider" the mapped table gets, in that for every field in the entire inheritance hierarchy, a column must exist in the mapped table. This may have undesirable consequence on the database size, since a wide or deep inheritance hierarchy will result in tables with many mostly-empty columns. |

|  |  |
| --- | --- |
| Joined Table Mapping | |
| DESC: | The Inheritance strategy “Joined” uses a different table for each class in the hierarchy. Each table only includes state declared in its class. Thus to load a subclass instance, implementation must read from the subclass table as well as the table of each ancestor class, up/down to the base entity class. |
| NOTE: | Using joined subclass tables is also called *vertical* inheritance mapping. |
| PROS: | 1. Using joined subclass tables results in the most *normalized* database schema, meaning the schema with the least spurious or redundant data. 2. As more subclasses are added to the data model over time, the only schema modification that needs to be made is the addition of corresponding subclass tables in the database (rather than having to change the structure of existing tables). 3. Relations to a base class using this strategy can be loaded through standard joins and can use standard foreign keys, as opposed to the machinations required to load polymorphic relations to table-per-class base types, described below. |
| CONS | Joined strategy is often the slowest of the inheritance models. Retrieving any subclass requires one or more database joins, and storing subclasses requires multiple INSERT or UPDATE statements. This is only the case when persistence operations are performed on subclasses; if most operations are performed on the least-derived persistent superclass, then this mapping is very fast. |

|  |  |
| --- | --- |
| Table Per Class | |
| DESC | Like the “Joined” strategy, the Inheritance strategy; “Table Per Class” uses a different table for each class in the hierarchy. Unlike the “Joined” strategy, however, each table includes all state for an instance of the corresponding class. Overall, as a result in most cases explicit loading is not necessary because objects are not loaded *lazily.* |
| PROS | The table-per-class strategy is very efficient when operating on instances of a known class. Under these conditions, the strategy never requires joining to superclass or subclass tables. Reads, joins, inserts, updates, and deletes are all efficient in the absence of polymorphic behavior. Also, as in the joined strategy, adding additional classes to the hierarchy does not require modifying existing class tables. |
| CONS | Polymorphic relations to non-leaf classes in a table-per-class hierarchy have many limitations. When the concrete subclass is not known, the related object could be in any of the subclass tables, making joins through the relation impossible. This ambiguity also affects identity lookups and queries; these operations require multiple SQL SELECTs (one for each possible subclass), or a complex UNION. |

**Data Security**

Known Threats and Exploits include:

1. Sensitive Data (threat)
2. Script Injection (exploit)

Combative Measures:

1. Threats
   1. Use an abstraction layer between User Interface and Database.
      1. Example: View Models in ASP.NET MVC
      2. Implement Authorization Constraints
   2. Limit Sensitive Information Requirements
      1. See Sensitive Information Definition section for examples
      2. Data deemed to be required will need to be encrypted
2. Exploit:
   1. Extension methods used in conjunction with the abstraction layer.
      1. Example: HTML encoding, Input Validation
   2. Limitation of services available to anonymous users.
   3. All Create, Read, Update, Delete (CRUD) features require a certain level of authorization. Known CRUD features that are available to affiliated Site Application Users will be closely monitored.
   4. **Quality Assurance** testing will need to be implemented pre-release.

**Sensitive Information Definition**

Sensitive information is data that must be protected from unauthorized access to safeguard the privacy or security of an individual or organization.

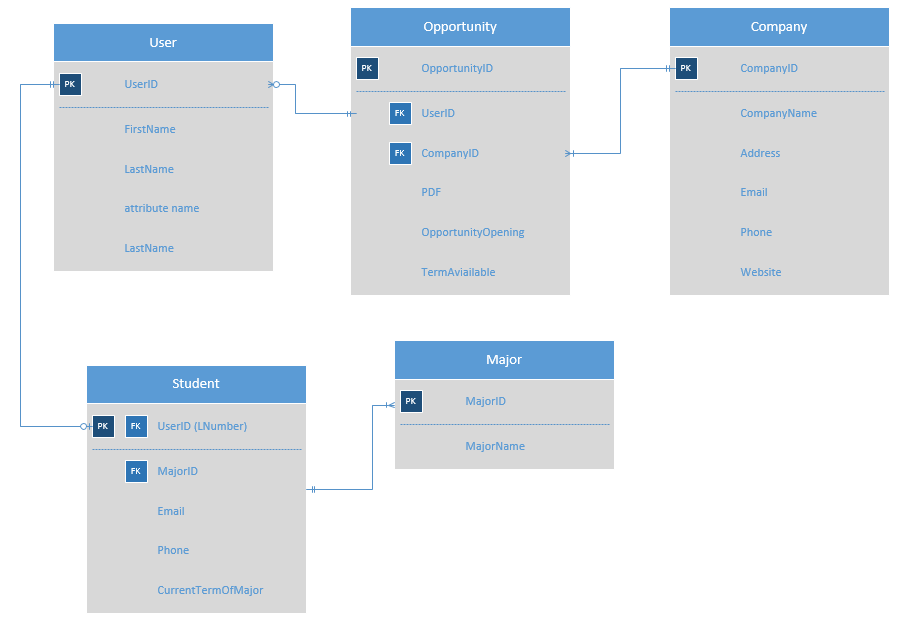
There are three main types of sensitive information:

**Personal information:** Sensitive personally identifiable information (PII) is data that can be traced back to an individual and that, if disclosed, could result in harm to that person. Such information includes biometric data, medical information, personally identifiable financial information (PIFI) and unique identifiers such as passport or Social Security numbers. Threats include not only crimes such as identity theft but also disclosure of personal information that the individual would prefer remained private. Sensitive PII should be encrypted both in transit and at rest. (Rouse & Wigmore, 2015)

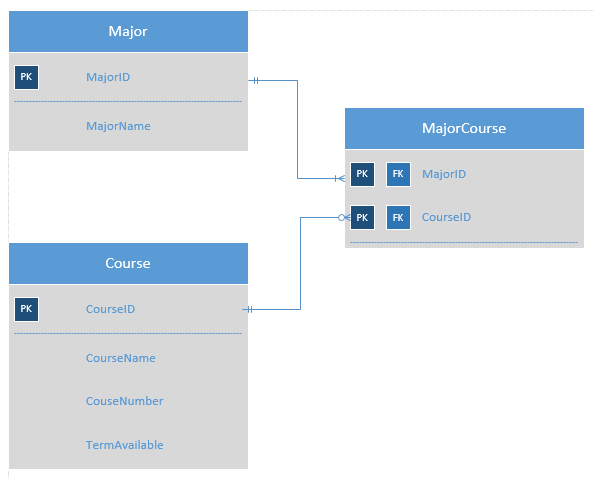
**Business information:** Sensitive business information includes anything that poses a risk to the company in question if discovered by a competitor or the general public. Such information includes trade secrets, acquisition plans, financial data and supplier and customer information, among other possibilities. With the ever-increasing amount of data generated by businesses, methods of protecting corporate information from unauthorized access are becoming integral to corporate security.  These methods include metadata management and document sanitization. (Rouse & Wigmore, 2015)

**Classified information:** Classified information pertains to a government body and is restricted according to level of sensitivity (for example, restricted, confidential, secret and top secret). Information is generally classified to protect security. Once the risk of harm has passed or decreased, classified information may be declassified and, possibly, made public. (Rouse & Wigmore, 2015)

**Database Schema**

****

**Database Schema Expansion**

****

# Works Cited

Rouse, M., & Wigmore, I. (2015, August 1). *What is sensitive information? - Definition from WhatIs.com*. (M. Rouse, I. Wigmore, Editors, & What Is; Tech Talk) Retrieved March 12, 2016, from http://whatis.techtarget.com/: http://whatis.techtarget.com/definition/sensitive-information