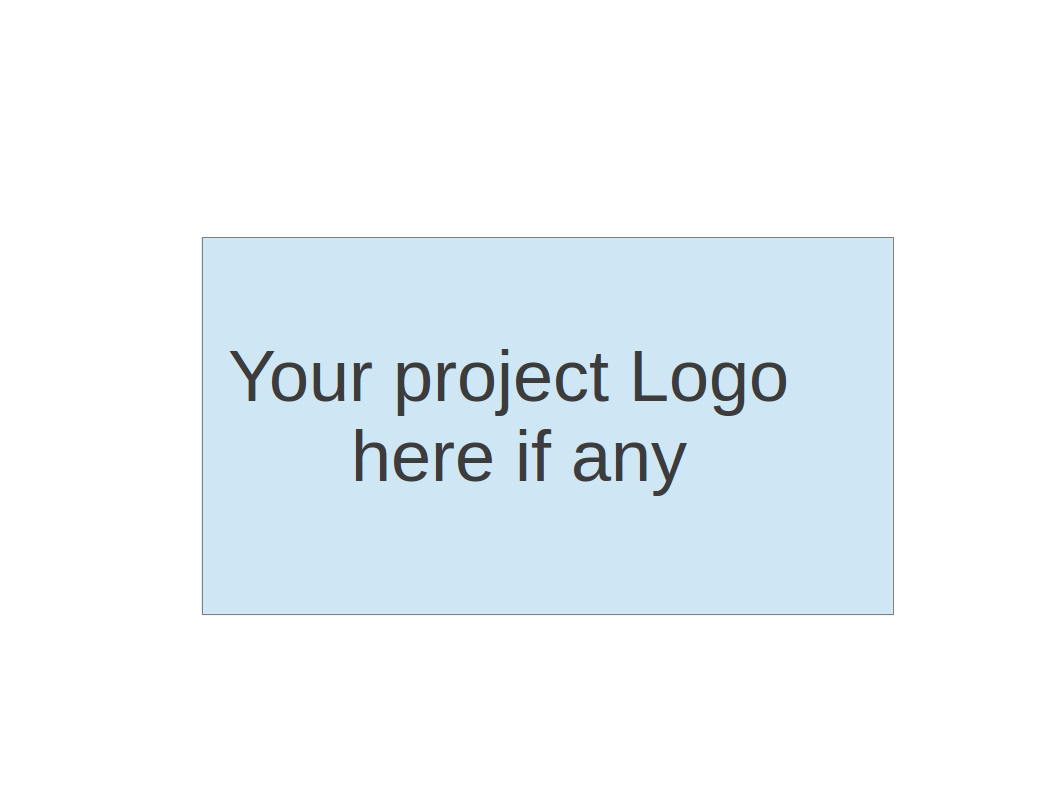
**CS673 Software Engineering** 

**Team 4 - Project Name**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Connor Richmond | Team Leader | *car* |  |
| Zak Kysar | Back up lead/Reqs | *zak* |  |
| Juan Sanchez | QA | *JPS* |  |
|  | Config |  |  |
| Rattikarn Dudley | Security | *Rk* |  |
| Molla Negash | Design | *MN* |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1** | **Car** | **10/1** | **Added project overview and algorithms section** |
| **2** | **JPS** | **10/2** | **Added ER and Relational Schema under design patters** |
| **3** | **CAR** | **11/13** | **Added UI screenshots and updated key algorithms section** |
| **4** | **CAR** | **12/12** | **Final cleanup** |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_buttcq9i221r)

[Design Patterns](#_x18fj36s1121)

[Key Algorithms](#_mtfbusfb0eq3)

[Classes and Methods](#_7ucksmkf6rzx)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Introduction

Our software system will provide search and archiving solutions for computer science based projects. We’re implementing a website that allows users to upload their own projects and search through those uploaded by others. The database structure will store all project info and authors. We will be implementing basic search functionality around project keywords that are manually input and also extracted from project titles and descriptions. All project information will be stored in a SQL Server database that will be viewable and searchable from project search and view pages. The system will be designed to be scalable to enable bug free project hosting as the number of projects stored increases moving forward.

# Software Architecture

*In this section, you will describe the decomposition of your software system, which include each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and class diagram for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. Database design should also be described if used.*

We are using a layered approach, the three core layers. Each layer provides a service to other layers. Each .cshtml file has these three layers. If a page does not require sql data, it will be a .html file, which only includes the first layer, presentation. It helps understand how our code is structured (and therefore know how to write and read them, and to describe how to implement them; we even use them in meetings, like, hey I made changes to the presentation layer of viewproject.cshtml file.). It provides an easy way of writing well-organized and testable code, which is why we have chosen it. It’s drawbacks are that it is less flexible and likely results in more lines of code, but that tradeoffs are worth it.

1. Presentation layer (Users Browser):

a. HTML using aspx pages

b. Java Script

c. Jquery

d. Css

2. Application Layer

a. ASP.NET,C# using aspx.cs pages

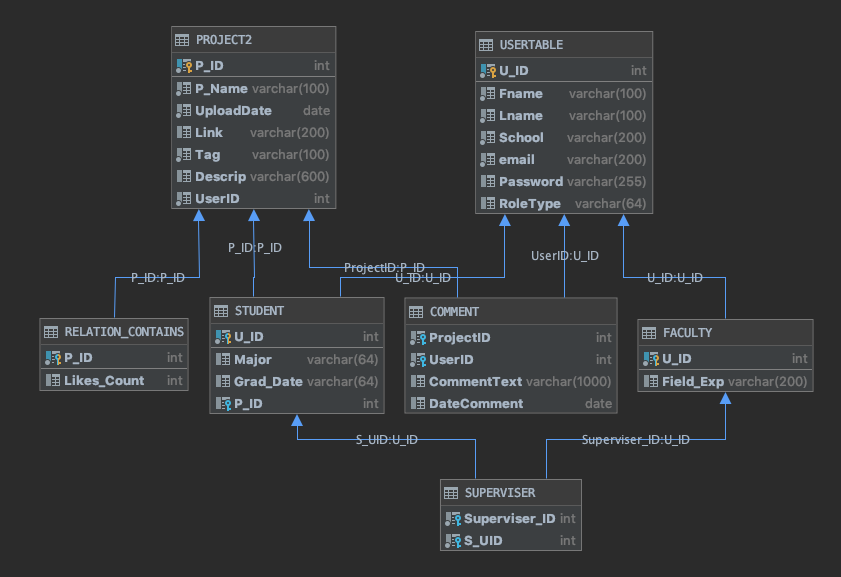
b. Communicates to presentation layer via URL Querying

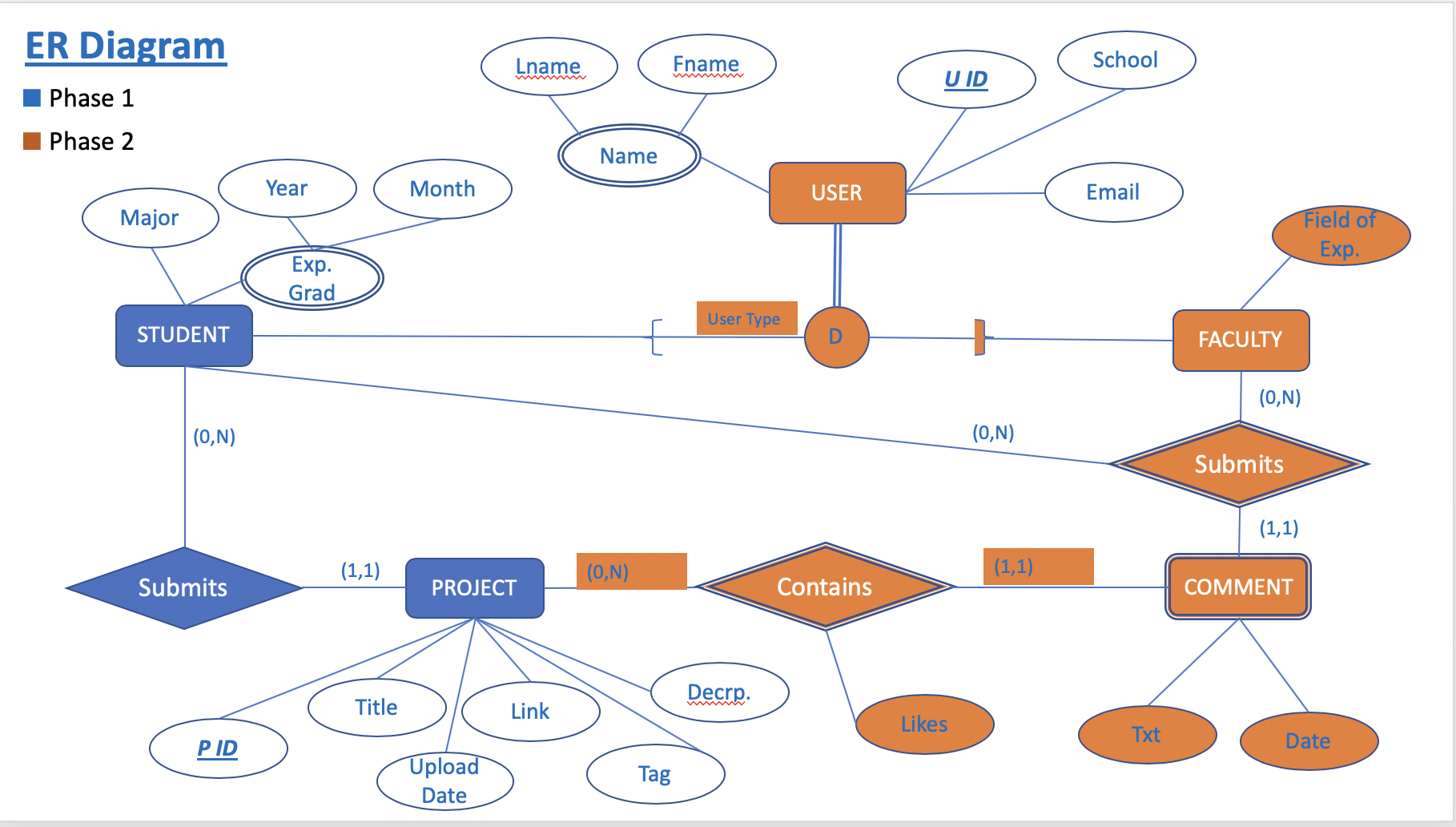
3. Data Layer

a. Sql server. Receives commands from Application Layer.

* + Components:
    - HTML Forms
    - SQL Server
    - CSS Styles
    - JavaScript navigation bar.

**Database Implementation:**

The team has implemented structural design patterns at the database design level to organize two different types of users, which can be students and faculty. Users will have their respective distinctive attributes at the subclass level (STUDENT and FACULTY), and common attributes at the super class level USER.



# Design Patterns

(In this section, you shall describe any design patterns used in your software system.)

The team has implemented a structural design pattern, the Facade pattern.

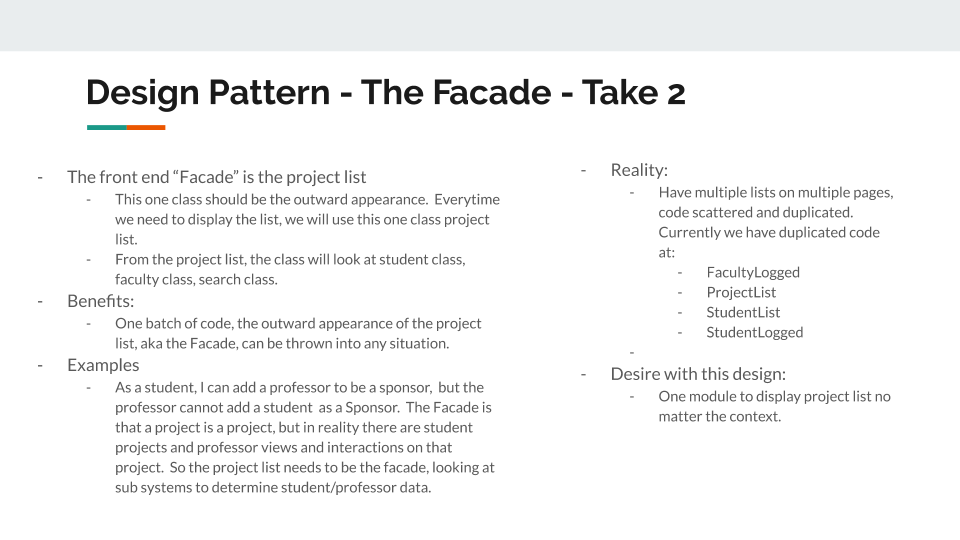
Facade’s definition is: “an outward appearance that is maintained to conceal a less pleasant or creditable reality.”

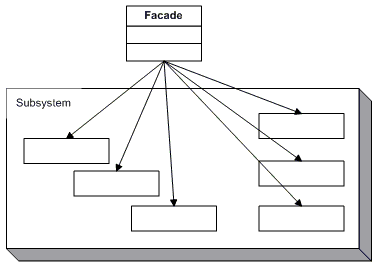
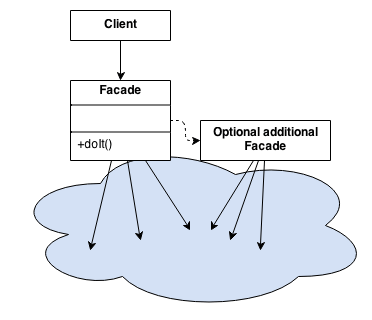
One of the reasons we need the Facade is because, in reality, we need to save a user that has different classes for Student and Staff. To the users, they will be users, which is the Facade, because in reality they could either be staff or students. This is needed, because they could eventually have different permissions at the project level.

Another reason for the Facade is that the projects will behave differently per user type. There will be different permissions on the project depending on what user is logged in. The FAcade is that the project looks like a simple project to the user, and the different behaviour is hidden, in a sense.

There will be two different Facade classes, one for user and one for projects.

Example: As a student, I can add a professor to be a sponsor. But the professor cannot add a student as a Sponsor. So the Facade is that that a project is a project, but in reality there are student projects and professor views and interactions on that project.



* References on Patterns:
  + <https://www.javatpoint.com/design-patterns-in-java>
  + <https://www.dofactory.com/net/design-patterns>
* After reading explanations from both of those sites for all patterns, I think we are Facade Pattern. References:
  + <http://jargon.js.org/_glossary/FACADE_PATTERN.md>
  + <https://www.javatpoint.com/facade-pattern>
    - Every Abstract Factory is a type of Facade method: <https://www.dofactory.com/net/abstract-factory-design-pattern>
  + <https://www.dofactory.com/net/facade-design-pattern>
    - 
  + <https://sourcemaking.com/design_patterns/facade>
    - 
    - 

# Key Algorithms

One of our main algorithms will be implementing keyword searching for uploaded projects. Returning relevant projects to the user’s search criteria will be crucial to the project portal’s success. There are 2 majors algorithms that we’ll need to implement proper keyword searching.

* Automated keyword tagging from project descriptions
  + Using simple NLP techniques to detect relevant search terms from user uploaded project descriptions.
  + By removing stop words we will extract relevant search terms that go beyond what users would upload manually.
  + First pass implementation will most likely consist of searching project description for known tags that have been manually input from other projects
  + Second pass can include a keyword extraction library perhaps this one built by npm:
    - <https://www.npmjs.com/package/keyword-extractor>
* Keyword search of currently stored projects
  + This is currently implemented at : <http://www.computerscienceprojectportal.com/submitsearch.cshtml>
  + The user can insert any string for each data point, and we’ll do a contains search on each data point. It’s not like a google search where one keyword will search across all data points, but that can be implemented in the future.
* Sorting:
  + By prioritizing keywords and potentially tracking views we can return the most relevant and trending projects to the user searching our database.

Pseudo code to come soon once database and add project functionality is implemented.

# Classes and Methods and Files/Pages (that are currently implemented)

* Home
  + Login
  + Forgot PW
* Registration
  + Signup
* ProjectList
  + View all projects
  + Search
  + View one Project
  + Comment on that one project
* StudentLogged
  + Self Project List
  + Edit
  + Delete
  + Log Out
* StudentProfile
  + Edit Profile
* AddProject
  + Only student adds project
* FacultyLogged
  + All Projects
  + Edit
  + Delete
  + Log Out
* FacultyProfile
  + Edit Profile
* StudentList
  + Faculty list of porjects
* User Interface (UI)

