Santa Clara University

COEN 174L: Software Engineering Lab

Group 5
Design Document

By

Giovanni Briggs

Justin Wong

Alex Hoff

October 12th, 2015

Contents

Introduction

Recently, students are having a harder time graduating in four years at colleges and universities across the country. Higher class enrollment creates more competition to get into classes and thus a greater need for the use of waitlists to track which student is next to enroll in a class should a seat open up for him or her.

Unfortunately, waitlists are often not as streamlined and efficient as they could be. At Santa Clara University, the original waitlist system required a student to e-mail a professor or department administrative assistant with his or her name, student ID, class section number and title, and the reason for addition to the waitlist. However, this system made a professor and an administrative assistant maintain two separate lists for the same course. Due to a lack of communication between the two individuals, no real structure or order existed, and students had no idea where they fell on a given waitlist. Professors were unable to accurately prioritize a waitlist because they did not have a complete picture of who was still waiting on a given class.

Santa Clara University's School of Engineering saw a need for change and instructed all engineering students to e-mail a department's administrative assistant when the student wanted to join a waitlist. This new system created one easily maintained list for each course, but also resulted in more traffic being directed towards the administrative assistants. Most engineering students do not seem to be aware of the current policy and still e-mail a professor directly requesting to join the waitlist for a course. Oftentimes, a student does not include the correct information in his or her request, resulting in more overhead for the administrative assistant as he or she has to e-mail the student back with further instructions. An administrative assistant is also responsible for managing the waitlists for multiple classes and sometimes misses a student's request to join. Cross-listed courses further complicate matters since each department's administrative assistant maintains his or her own waitlist for the same course.

We propose a website that will allow a student to join a waitlist for a particular class. A student can fill out a form with all necessary information including name, e-mail, SCU ID number, the class he or she wants to get into, and why he or she needs to be in that class. An administrative assistant will have the opportunity to access the waitlist and a professor will be able to prioritize the waitlists for his or her classes. This system will also provide students with feedback about their spots on the waitlist so they will know in advance if they should make other plans. We intend to make the system easily configurable so that it requires low levels of maintenance while also providing an easy to use interface so that students can quickly join a waitlist and administrative assistants can focus on placing students into classes.

Requirements

Based on our understanding of the problem, these are the functional and nnm-functional requirements along with the expected design constraints.

Functional Requirements

- Critical
  - The system will be web based
  - The system will keep a waitlist in a database that can only be accessed by an admin
  - The system will keep one list for cross listed classes
  - The system will allow students to add themselves to a waitlist
  - The system will guarantee that the proper information is provided for each waitlist request
  - The system will allow admins to change students priorities
- Recommended
  - The system will allow students to see what position they are on the waitlist
- Suggested
  - The system will allow students to remove themselves from the waitlist
  - The system will allow admins to remove students from the waitlist

Non-functional requirements

- The system will be fast and easy to use for both students and admins
- The system will be secure
- The system will be aesthetically appealing

Design Constraints

- Run on Engineering Computing Center machines

Use Cases

From the requirements and our understanding of the problem, we have identified three different classes of users: students, professors and department administrative assistants, and system administrators. As shown in Figure 1, each of these groups will interact with the system in different ways.

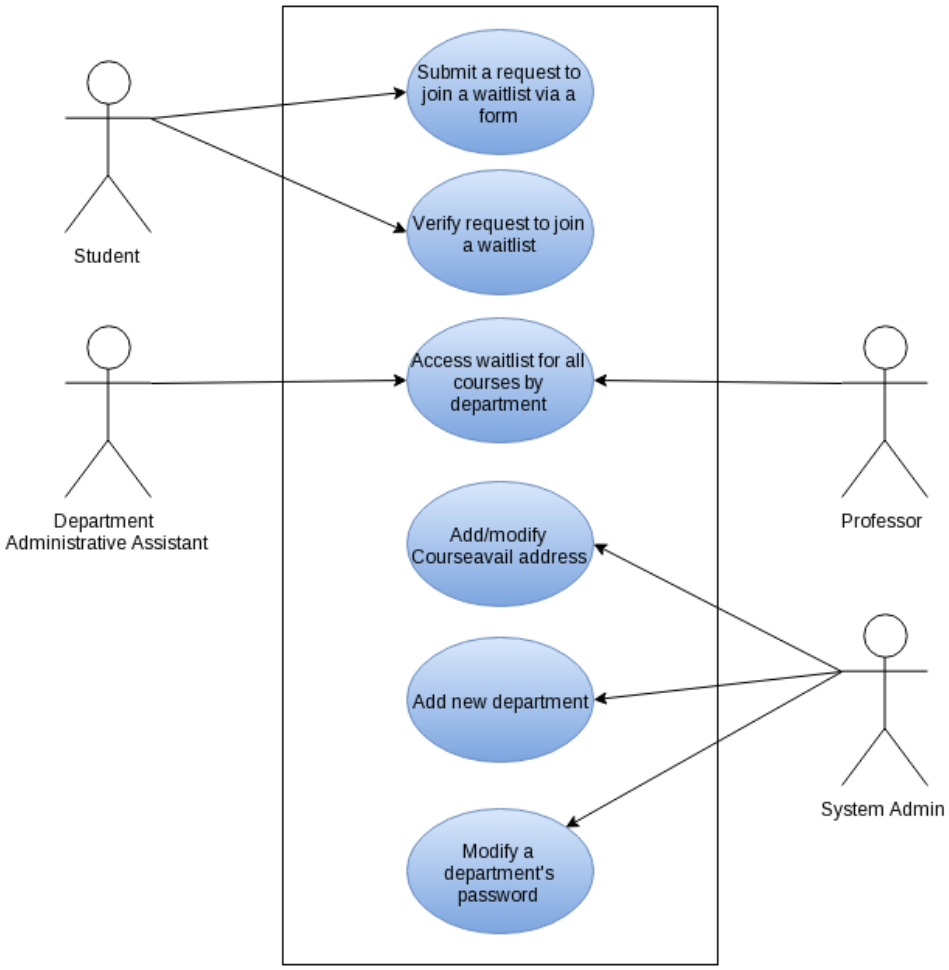

Figure 1: Use Cases

Submit request to join a waitlist via a form

Actor: Student

Goal: Add student to specified waitlist

Preconditions:

1. The class exists

Postconditions:

1. N/A

Steps:

1. Student accesses form via web browser
2. Student fills out each field
    a. Department offering course
    b. Course's Name/Section
    c. Student's Name

d. Student's ID
e. Student's e-mail
f. Reason why student needs to be in class

3. Student press submit button to send his or her request to join a waitlist

Exceptions:

1. All fields in form do not contain valid data. The system will notify the user and ask them to try again.

Verify Request to Join

Actor: Student

Goal: Verify that the student actually submitted the request to join and then add him or her to the waitlist

Preconditions:

1. Someone requested to add the user with the specified e-mail to a waitlist
2. Student received confirmation e-mail

Postconditions:

1. New entry contained in specified waitlist

Steps:

1. User opens e-mail
2. User clicks on verification link
3. The waitlists are divided by quarter and year. There will be one link for each
    a. Example: 2015 Fall

Exceptions: N/A

Access Waitlist for all courses offered by a department

Actor: Administrative assistant or Professor

Goal: Download waitlist for department

Preconditions:

1. The department has login credentials for the system

Postconditions:

1. N/A

Steps:

1. User logins in using his or her department's username and password
2. User clicks on link to download the appropriate waitlist
    a. The waitlists are divided by quarter and year. There will be one link for each

i. Example: 2015 Fall quarter

Exceptions: N/A

Modify Courseavail address

Actor: System admin

Goal: Update the address that the server uses to communicate with Courseavail in order to keep communication alive.

Preconditions:

1. The Courseavail server changes address and our system can no longer communicate with it

Postconditions:

1. System is communicating with Courseavail

Steps:

1. User goes to *admin* page
2. User logs in using correct credentials
3. User clicks on *Update Courseavail Address.*
4. User types new server address into field
5. User clicks *Submit* to have the system update the address it uses to try and communicate with Courseavail

Exceptions: N/A

Add New Department

Actor: System admin

Goal: Add a new department login and waitlist group to the system

Preconditions:

1. There is a new department that wants to be included in the system
2. The department doesn't already exist in the system

Postconditions:

1. System contains login information for the department

Steps:

1. User goes to *admin* page
2. User logs in using correct credentials
3. User clicks on *Add New Department*

4. User enters Department's:
    a. Name
    b. Login username
    c. Login password
5. User clicks *Submit* to create new login credentials for a department

Exceptions: N/A

Modify a Department's Password

Actor: System admin

Goal: Change a department's password to access the system.

Preconditions:

1. The department already has login credentials in the system

Postconditions:

1. System contains updated login information for the department

Steps:

1. User goes to *admin* page
2. User logs in using correct credentials
3. User clicks on *Change Department Password*
4. User enters Department's:
    a. Name
    b. Username
    c. Current password
    d. New password
5. User clicks *Submit* to have the system update the department's password

Exceptions: N/A

Activity Diagram

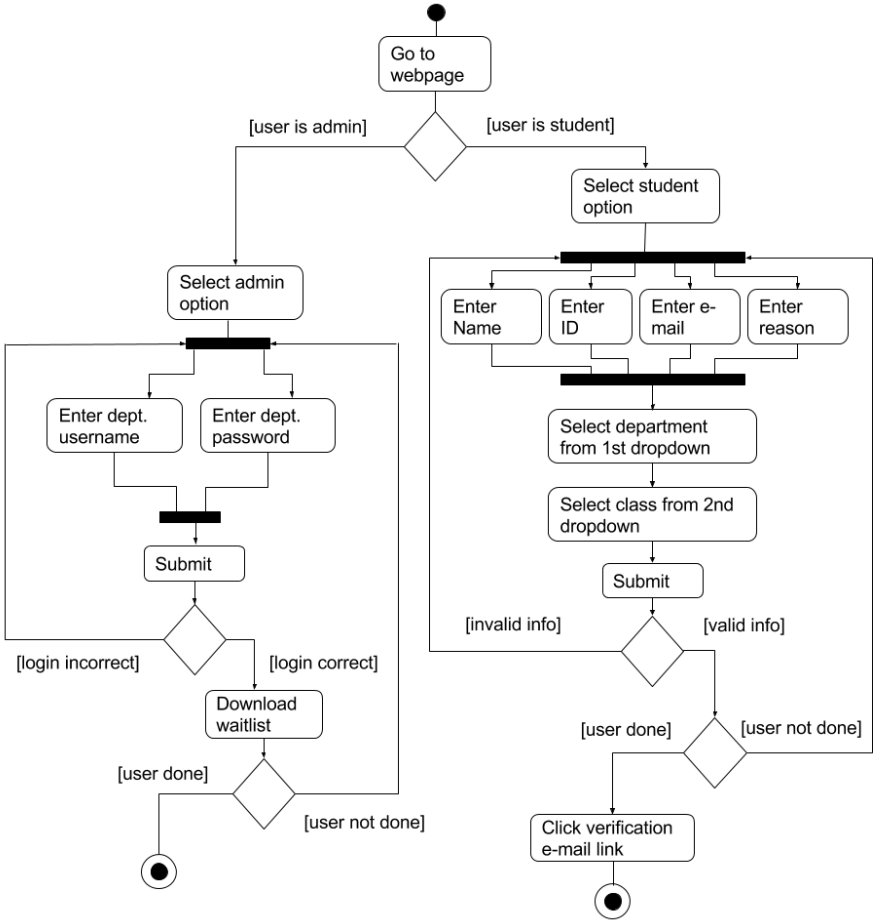

Figure 2: Activity Diagram

Conceptual Model

Student users will first be presented with a form much like the one in Figure 3. They will be able to input their credentials and then choose which class section they would like to be added to using the two drop down menus. The second drop down menu will populate based on the department chosen in the first drop down. Students will then give a short reason for why they would like to add the class and then submit the form which will then be checked to ensure that the personal information is correct. The completed form will then be sent via email to the student where they will click on a link in the email to confirm the validity of the form. The student will finally be added to the appropriate waitlist which will display all of their information for the admin to decide on priority.

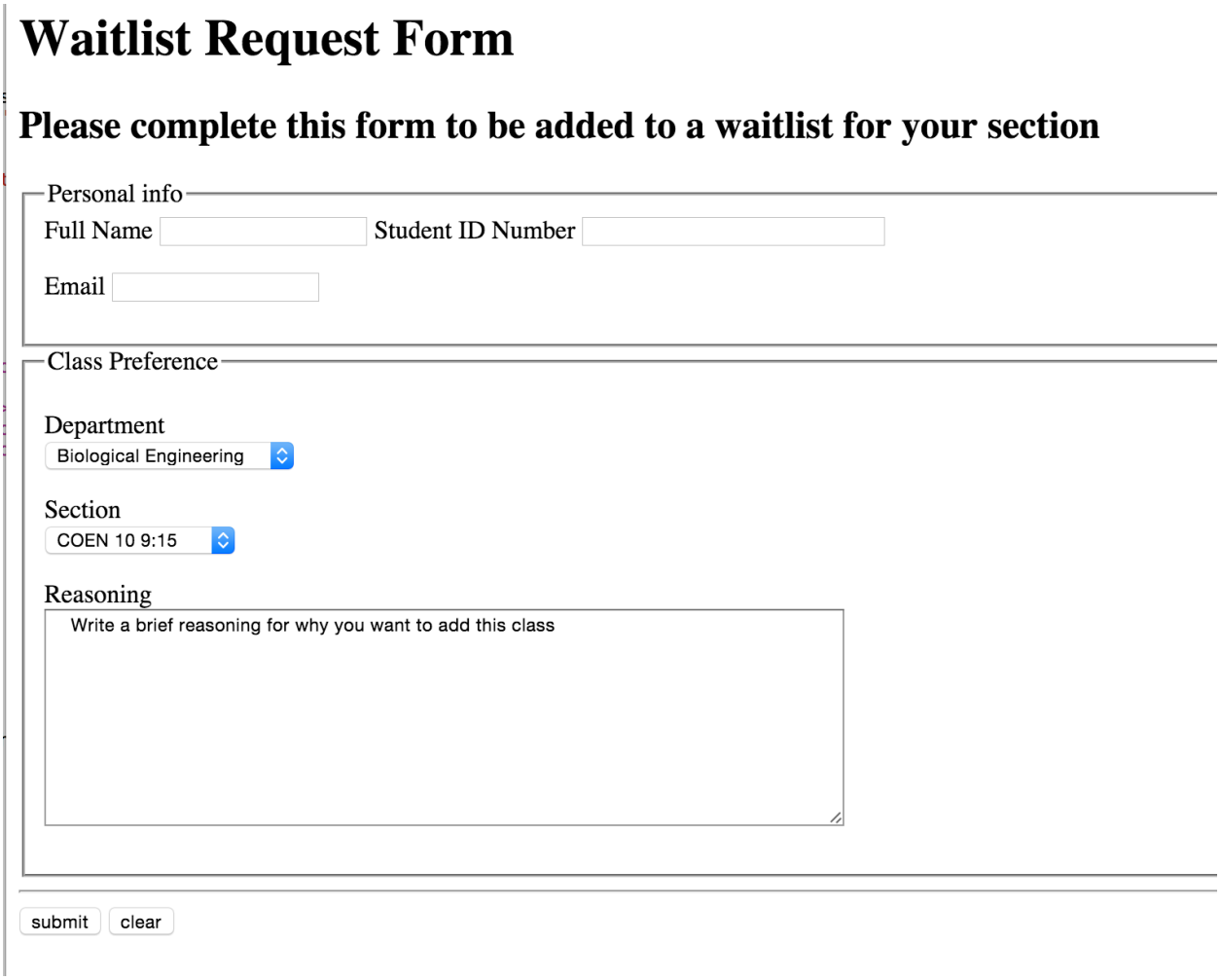

Figure 3: Form to request to join a waitlist

The admin will log into the waitlist using their department's credentials (See Figure 4).

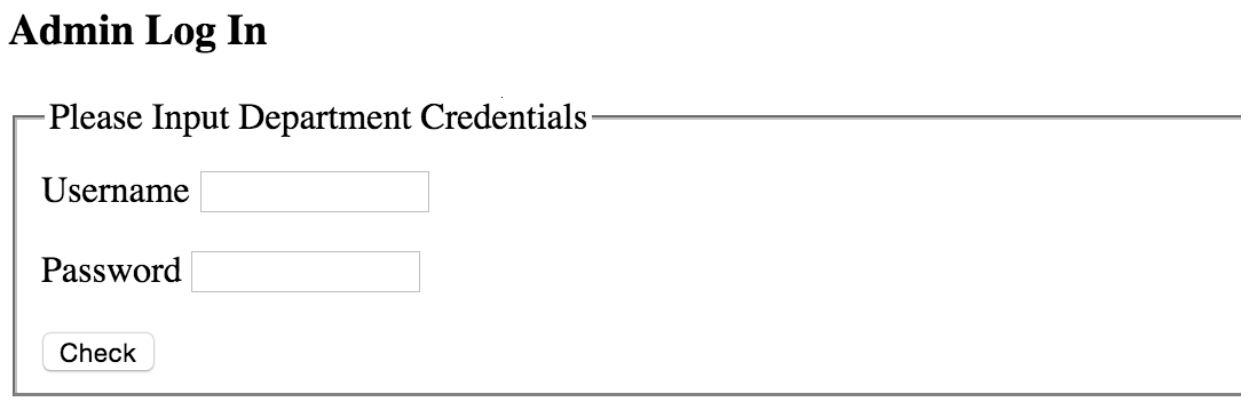

Figure 4: Department Administrative Assistant and Professor login form

The admin will then be presented with a table listing the waitlists for all class sections in their department with the students names, emails, years, and reasoning for adding the class (See Figure 5). The admins can then edit a student's priority moving them up or down the wait list or remove students they do not deem necessary to be in the waitlist or send permission numbers to students they wish to add.

Priority	Name	Email	Year	Edit Priority
1	Jim Doe	jdoe@scu.edu	Senior	EDIT
2	Sally Sue	ssue@scu.edu	Senior	EDIT
3	Dan Smith	dsmith@scu.edu	Junior	EDIT
4	Average Joe	ajoe@scu.edu	Senior	EDIT
5	Timmy Brown	tbrown@scu.edu	Sophomore	EDIT
6	Randal Grey	rgrey@scu.edu	Junior	EDIT

Figure 5: Web table view of waitlist

Technologies Used

Based on the problem at hand and our own knowledge of web programming languages, we decided to use HTML, CSS, and Javascript to create the form presented to the user as well as Bootstrap to streamline the entire process. To communicate with the server we decided to use PHP.

- HTML
  - A markup language that defines the basic structure of a webpage.
- CSS
  - A style sheet language used to separate the presentation of document from its structure
- Javascript
  - A high-level programming language generally used to add interactivity to webpages.
- Bootstrap

    - HTML, CSS, and Javascript framework designed for rapid web development
- PHP
    - High-level programming language used for web development that executes on the server-side.

Architectural Design

Our system implements a client-server model that communicates with a design center machine which handles all the back end processes using a data-centric architecture to store data submitted by clients. Clients will be presented with an HTML webpage styled with CSS and using Javascript functions to add functionality with PHP to communicate with the server (See Figure 6). When a user submits data it will be passed from the client to the server which will then be updated. When a user requests data, the server will pass the data to the client using https to authenticate the user.

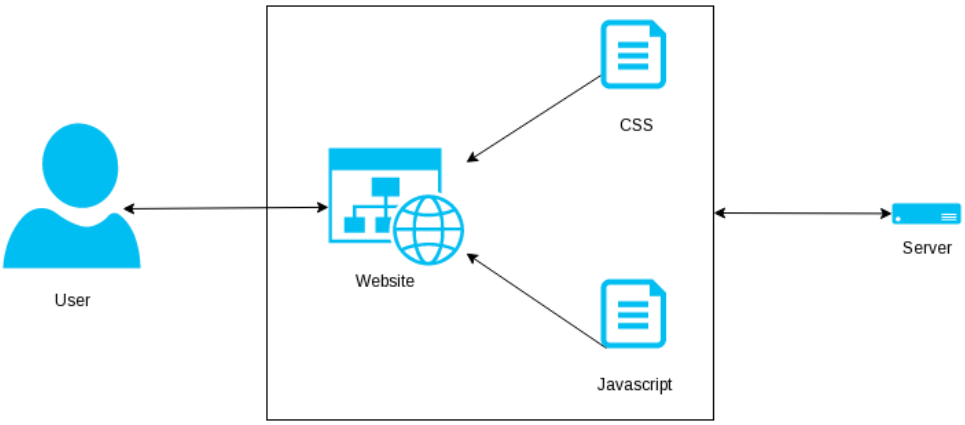

Figure 6: Architecture Design

Design Rationale

We designed our system in order to meet all functional requirements while keeping the system easy to use and maintain. We have identified three different classes of users, and our web-based solution will have specific web-pages geared to each class. This will allow each class of user to quickly access the information that they need. The use of e-mail verification allows us to implement some form of authentication without having to interface directly with any of Santa Clara University's various authentication systems. Using Courseavail makes the setup of the system easier, since all a system admin needs to do is point the server to the correct Courseavail address. The system will then automatically pull in class information so departments don't need to enter their course offerings in multiple places. This method has been proven to work by the SCU Classes (http://www.scuclasses.com)

website which is also consistently screen scraping the data from Courseavail. Doing this ensures that the system has minimal overhead while still providing the essential functionality.

Justification for technologies used

- **HTML** is necessary for creating web-pages.
- **CSS** will allow us to design clean, easy to use and appealing web pages.
- **Javascript** will allow the system to make the pages dynamic and interactive. It will also be extremely useful for doing data validation for the request-to-join-waitlist form.
- **Bootstrap** is an open source framework that will help us quickly design pages by leveraging the work of others.
- **PHP** will give the system the server side functionality that it needs.

Test Plan

General usability

1. Test all pages on different operating systems with different browsers to make sure that the site is truly cross platform

Form Functionality

1. Check form validation functions by submitting faulty requests
2. Submit valid requests and make sure that user is directed to appropriate pages afterwards

Waitlist Access

1. Test login by supplying correct and incorrect credentials
2. Test download functionality by attempting to download available waitlists
3. Test waitlist viewing functionality by attempting to load the waitlist in browser and make sure that the browser displays all available data for that waitlist

Waitlist generation

1. Submit requests through form, validate the requests and then check the waitlist to see if all information went through correctly

Verification E-mails

1. After submitting a form, check to see if server sent corresponding confirmation e-mail
2. Check to make sure confirmation e-mail link is functional by clicking on link and then checking waitlist to see if data is there

Risk Analysis

Given our design architecture and the functionality we aim to provide, these are the risks that we predict with our development.

Risks	Consequences	Probability	Severity	Impact	Mitigation Strategy
Incorrect Requirements	Product is not completed as the customer wished	.4	10	4	-Constantly communicate with customer about questions about the project -Have the customer test the project throughout development
Poor Communication	Bad chemistry between the team, working on the same functionalities, slow development	.2	10	2	-Create group message so all group members are in the know at all times -Create group hierarchy
Insufficient Technical Ability	Group members forced to spend large amounts of time learning new skills resulting in slow development	.2	8	1.6	-Communicate within the group who has what skills -identify necessary new skills early in development so they can be learned ahead of time
Too much functionality	Unable to complete the project on time	.2	8	1.6	-Design the project with reasonable functionalities -Create a timeline to finish functionalities on time
Illness/Heavy homework load	Teammate unable to help with completion of project	.3	3	.9	-Schedule time wisely so appropriate time for sleep can be given -eat a healthy diet

Development Timeline

See Appendix A

Conclusion

Due to the requirements of this project, we believe that our implementation will be done in the most effective manner possible. We hope to constantly communicate with the customer to make sure all necessary functionalities are available and our finished product exceeds expectations.