Project Notebook: Course Registration

CS 532: Software Engineering

Group Members:

Reese Hartwell, Austin Frey, Caleb Greenfield,

Maxim Kanayasami, Connor Reid, Mariel Reyes

2nd Deliverable: 10/20/22

Test Plan, project architecture, data model presentation, detailed design, and updated project plan in your Project Notebook.

Complete sections 7, 8, & 9.

Update 4, 5, 6 & 11 as annotated in the Project Notebook template.

Signatures

Reese Hartwell

Austin Frey

Mariel Reyes

Connor Reid

Caleb Greenfield

Maxim Kanayasami

Section 3:

a.) Roles and Responsibilities:

- Project Manager: Reese Hartwell
- Requirements Manager: Caleb Greenfield
- Design Manager: Austin Frey
- Software Manager: Austin Frey
- Software Developer: Maxim Kanayasami, Reese Hartwell, Caleb Greenfield, Austin Frey, Connor Reid, Mariel Reyes
- Test Manager: Connor Reid
- Configuration Manager: Mariel Reyes

b.) Development Environment:

I. MacOS/Windows
II. PHP, HTML, CSS, JavaScript
III. MAMP, MySQL, phpMyAdmin
IV. Simplicity and ease of configuration between files. PHP will allow us to seamlessly tie frontend and backend together without having to use extra languages. HTML and CSS are fundamental in creating an appealing web server. Bootstrap is a framework that will make the UI of our webserver much easier and more intuitive to work with. MySQL is being used because our team has some experience with it as well as the fact that it is a relational database which is important in the assignment we have chosen. Some specific functionalities

c.) ICSM Common Case:

Case 1 → Software-intensive application or system

Using the ICSM Common Cases table, we concluded that our project does not fall under any cases 2-6. It's software-intensive for a couple reasons. We are developing a standalone software system and are attempting to implement a feature in which we allow multiple users to access the web server and work independently at the same time. Ideally, we will want multiple users to be accessing multiple databases and reports in tandem which would result in a higher performing software

d.) Methodologies/Techniques to be used for Development/Rationale:

We have decided to utilize Agile methodologies with an iterative approach. Specifically, we used the SCRUM framework and eXtreme Programming (XP). We used SCRUM by meeting 2-3 times a week discussing what we have done in the last few days, what we are currently working on, and what we still need to do by the next sprint/iteration. We also used XP in several aspects. Primarily, we used XP's ideology of the simplest approach in both design and code. We wanted to focus on our functional requirements so we took a very basic and simple approach just

to get the requirements up and running before we added special elements of our own. We also used pair programming. Once we established the foundations of each student/teacher/admin web pages, we broke off in teams of two for their specific functionality. Additionally, throughout development, we continuously integrated our changes in an iterative fashion.

Section 4.)

a.) Software Size:

AP COUNTING RULES FOR *SCREENS*

# and source of data tables → **# of views contained ↓**	**Total <4 (<2 server, <3 client)**	**Total <8 (2-3 server, 3-5 client)**	**Total 8+ (>3 server, >5 client)**
<3	**Simple**	**Simple**	**Medium**
3-7	**Simple**	**Medium**	**Difficult**
8+	**Medium**	**Difficult**	**Difficult**

==========

(# data tables, #views contained)

SignMeUp = <4, 3-7 (Simple)
Electronic Student Record = <4,<3 (Simple)
Course Registration = <4, 3-7 (Simple)
Major Course Requirements = <8, 3-7 (Medium)
Faculty and Course Information = <8, 3-7 (Medium)
Course Grades = <8, <3 (Simple)

==========

AP COUNTING RULES FOR *REPORTS*

# and source of data tables → **# of sections contained ↓**	**Total <4 (<2 server, <3 client)**	**Total <8 (2-3 server, 3-5 client)**	**Total 8+ (>3 server, >5 client)**
0-1	**Simple**	**Simple**	**Medium**
2 or 3	**Simple**	**Medium**	**Difficult**
4+	**Medium**	**Difficult**	**Difficult**

==========

(#data tables, # sections contained)

SignMeUp = <4, 1 (Simple)
Electronic Student Record = <4, 2 (Simple)
Course Registration = <4, 2 (Simple)
Major Course Requirements = <8, 3 (Medium)
Faculty and Course Information = <8, 3 (Medium)
Course Grades = <8, 3 (Medium)

==========

AP COUNTING RULES *WEIGHTING*

Complexity weight → **Elements type ↓**	**Simple**	**Medium**	**Difficult**
Screens	1	2	3
Reports	2	5	8
3GL/4GL Components			10

 # Application Points:

- **Screens: 4 Simple, 2 Medium = 4 + 4 = 8**
- **Reports: 3 Simple, 3 Medium = 6 + 15 = 21**
- **3/4GLs: 26, 6 Difficult = 60**

89 Application Points

 # New Application Points = (APs)(100-%reuse)/100
(89)(100-40)/100 = 53.4
Productivity Rate, PROD = NAP/person-month
PM = NAP/PROD → 53.4/6 = 8.9
Therefore, with 6 people, it would take an average of 1.5 months to complete the project

b.) Schedule:
Schedule Link

Section 5.)
Estimate of Labor Hours:

OVER THE COURSE OF 10 WEEKS:

GROUP TIME:

40 hours group meetings x 6 people = **240 hours**

INDIVIDUAL TIME:

3 hours/week x 6 people = **180 hours**

TOTAL ESTIMATED HOURS:

420 hours

Section 6.)

a.) Statement of Scope:

i. Overall Functionality

The overall functionality provided by this software will be to connect students and administrators in one place that has all the tools needed for their roles when it comes to course registration. Administrators would have access to all of their courses and be able to make any changes to the grade information in any of their courses. Students would be able to register for all of their courses that are in accordance with all of the information in their account that would affect their ability to register for a course.

All users will have the ability to see a list of all required courses for a certain major. While only major advisors, and the student the information is pertaining to, may see a list of completed courses, the history of their enrollment, and their progress in their major outline. Users will also be able to generate statistics on courses, departments, and the university. Along with lists and short descriptions of any faculty member or courses within that university and/or department.

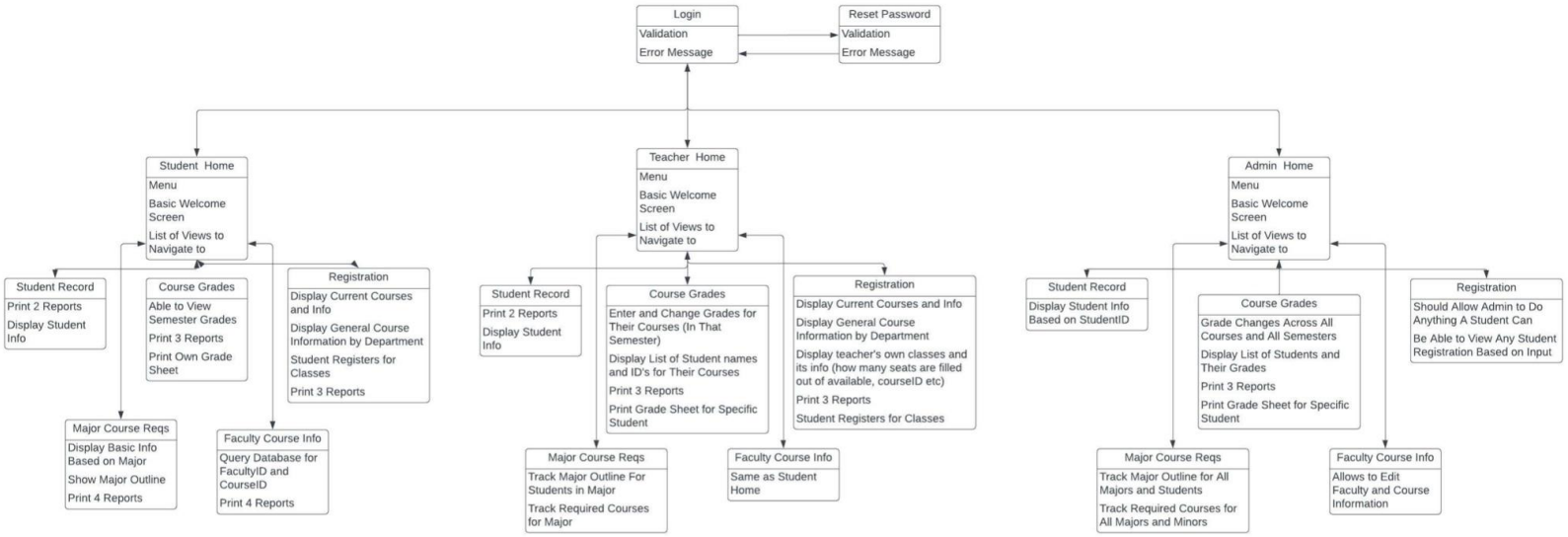
ii. Performance Requirements

The performance of this software should be efficient and work on any modern computer. When registering for courses on the course registration aspect of the website, any transaction should warrant a five seconds or less response time from the website. This website should also have the ability to handle multiple users simultaneously without any issues. In order for those standards to be met the software should possess a proper run time for every function.

The website should also have a secure system with object based security. This is so that the clientele will have the reassurance that their information is secure when using our website. We also want the size of the website to stay under

800 kb and to provide little to no errors for the users. A smaller sized website with a lower complexity will be easier to maintain and fix any bugs that may come along the way.

iii. UML Diagram



b.) Discussion of Requirements Analysis:

i. Table

[Requirements Table Link]

ii. Requirements Discussion

Discussion of completeness, consistency, testability, etc. of requirements and any changes that were required to address these types of issues.

Significant progress has been made since our project's conception. Our student pages are our most fleshed out and complete as we have spent the most time on it. No student-specific functionalities are missing only quality of life elements that would enhance user experience. Additionally, in terms of teacher-specific functionalities, we have given teachers permission to change all students' grades rather than grades for students only in their classes. Other than that, the teacher is almost entirely complete. For admin-specific functionality, we are missing a few key functionalities including adding course and student schedules, and adding major requirements. We have actively been applying blackbox testing techniques, testing our functional and nonfunctional

requirements and more importantly database testing and integration testing. Since our projects conception we have actively been testing, ensuring there are minimal conflicts as we move forward.

iii. Quantitative Summary:

Our course registration project is divided into 7 functional areas each with their own requirements. These include: *General* (29 requirements), *SignMeUp Framework* (6 requirements), *Electronic Student Record Subsystem* (5 requirements), *Course Registration Subsystem* (8 requirements), *Major Course Requirements Subsystem* (6 requirements), *Faculty and Course Information Subsystem* (5 requirements), *Course Grades Subsystem* (7 requirements). This is a total of 64 requirements for our project that will be addressed going forward in top-level design. These requirements were gathered directly from the project description document. There were small changes to the requirements that we had to address due to certain risk management issues and time constraints. These requirements were mainly from the admin functionality and a student/teacher/admin requirement that is used in each section (individual student's major outline).

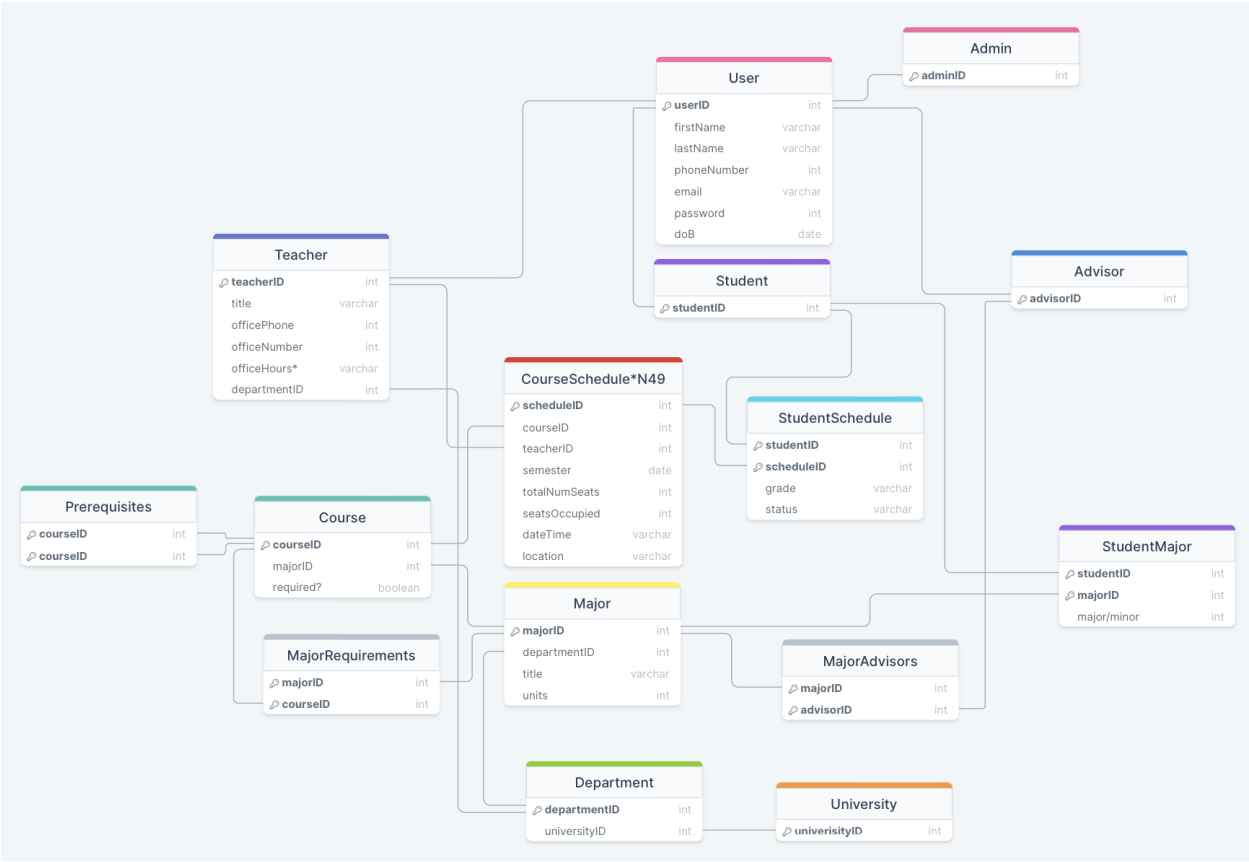
Section 7.)

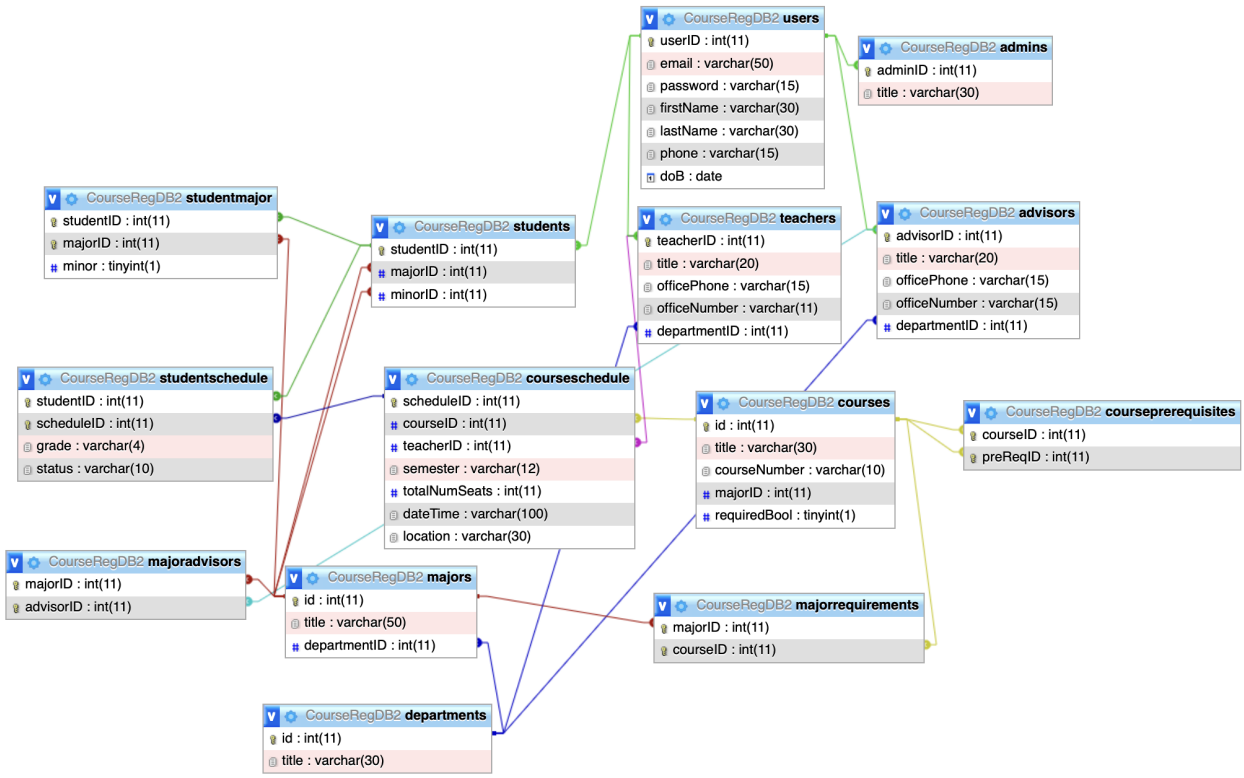
a.) Top-level architecture diagram:

Our course registration system is a heterogeneous system. Primarily an object-oriented system, we have also decided to incorporate a database-centric architecture side too. We decided to include database-centric architecture because as we developed more of our functionality, we began to realize that our system is very database-heavy. All of our key logic involved querying the database and using sql-specific commands in PHP code. We also opted for object-oriented architecture because we have many "classes" and objects. Generally the objects are broken down into a student object, a teacher object and an admin object, each representing a user. The objects each have their own respective attributes. Each object does not depend on another object but they can communicate, pass messages, and reflect changes one object can have on another. We represented our architecture using what our book (*Essentials of Software Engineering*) describes object-oriented as UML and use case diagrams.

Lucidchart link

b.) Database Design (ERD):

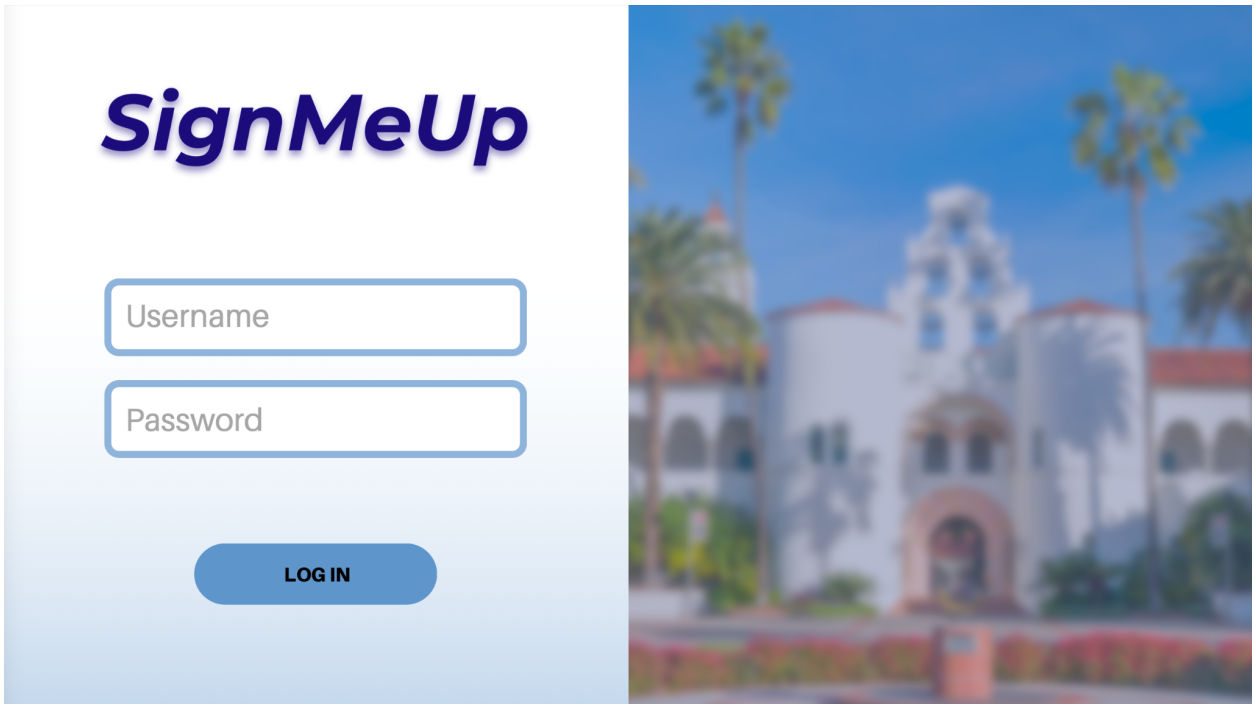


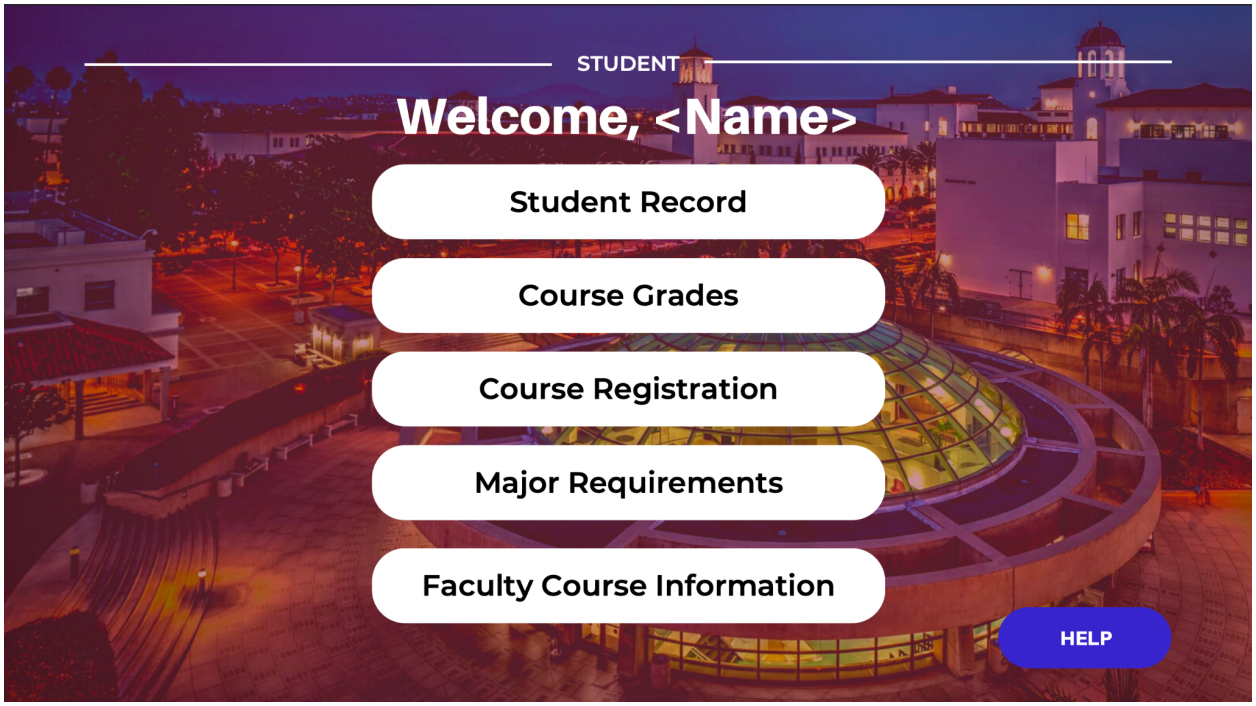


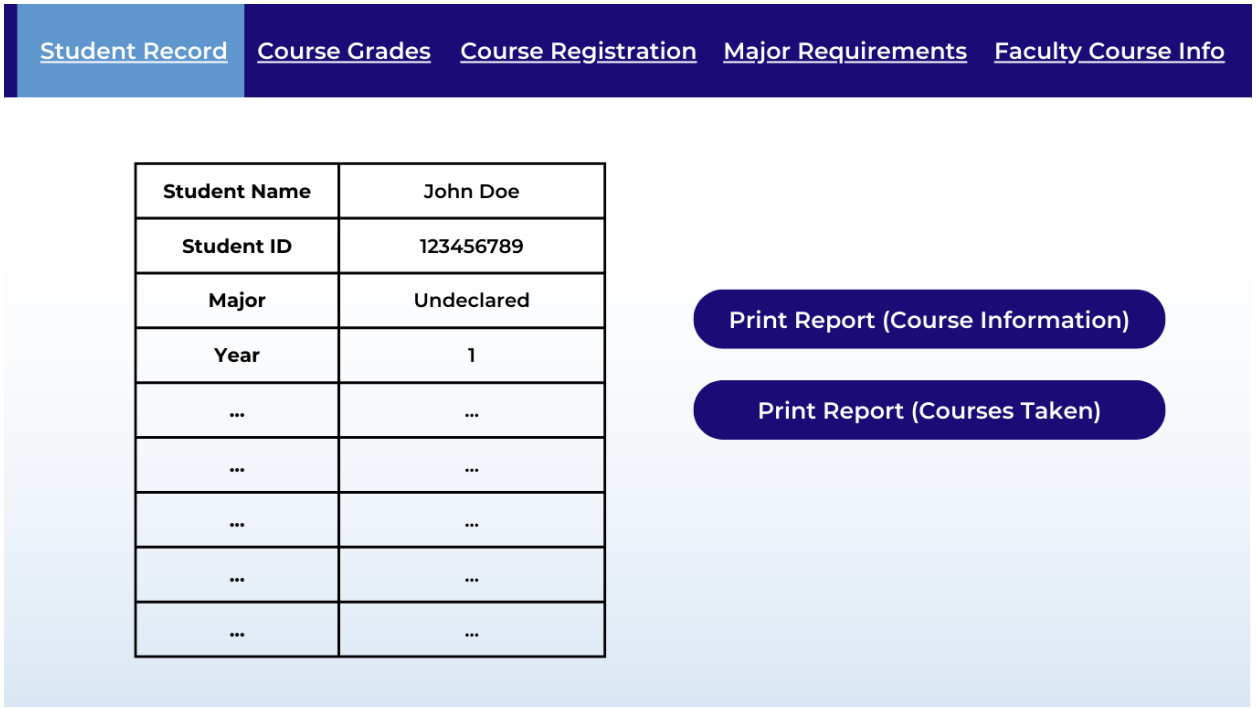
c.) Requirements mapping to components:

[Requirements Table Link]

d.) UI prototype:







e.) Detailed Design:

1. STUDENT HOMEPAGE
    a. Student Record (ER):
    Displays: Specific personal student information, and list of courses by semester
    Navbar/Menu: Will direct user to other subsystems
    Buttons: 1 Logout button and (X) Drop buttons for enrolled courses

b. Course Registration (REG):
   Displays: 2 Displays: 1 Display, a result of querying DB and a display when 'Register' button is pressed
   Navbar/Menu: Will direct user to other subsystems
   Buttons:1 Submit query button, 1 Logout button, (X) register buttons for each course
c. Major Course Requirements (MAJOR):
   Displays: Query DB to display requirements for specific majors
   Navbar/Menu: Will direct user to other subsystems
   Buttons: 1 Logout button, 1 Sumbit query button
d. Faculty and Course Information (FCI):
   Displays: Complex view describing instructor, course, and semester relationships depending on query
   Navbar/Menu: Will direct user to other subsystems
   Search Bar / Input Bar: 2 for queries relating to FacultyID and CourseID
   Buttons: 1 Submit query button, 1 Logout button
e. Course Grades (GRADE):
   Displays: 2 Displays, one for querying grades for one semester, one for grade sheet showing all courses, GPA, and credits earned
   Navbar/Menu: Will direct user to other subsystems
   Buttons: 1 Logout button, 1 Submit query button

2. TEACHER HOMEPAGE
   a. Student Record (ER):
      Displays: 2 major displays. One showing list of students, one being another page which shows student record (identical to student's view) for specific student
      Navbar/Menu: Will direct user to other subsystems
      Buttons: 1 Logout button, one button for each student listed redirecting teacher to new page when clicked
   b. Course Registration (REG):
      Not a feature in Teacher view
   c. Major Course Requirements (MAJOR):
      Not a feature in Teacher view
   d. Faculty and Course Information (FCI):
      Displays: Complex view describing instructor, course, and semester relationships depending on query
      Navbar/Menu: Will direct user to other subsystems
      Search Bar / Input Bar: 2 for queries relating to FacultyID and CourseID
      Buttons: 1 Submit query button, 1 Logout button
   e. Course Grades (GRADE):

Displays: 2 Major displays. One being a list of students. Another being a grade sheet of selected student which allows a teacher to change their grades,
Navbar/Menu: Will direct user to other subsystems
Buttons: 1 Logout button, 1 back button, one button for each student

3. ADMIN HOMEPAGE
    a. Student Record (ER):
    Displays: 2 major displays. One showing list of students, one being another page which shows student record (identical to student's view) for specific student
    Navbar/Menu: Will direct user to other subsystems
    Buttons: 1 Logout button, one button for each student listed redirecting teacher to new page when clicked
    b. Course Registration (REG):
    Displays: 1 display allowing an admin to insert a new course into DB
    Navbar/Menu: Will direct user to other subsystems
    Buttons: 1 Submit query button, 1 Logout button
    Fields: 6 fields for each attribute in course table
    c. Major Course Requirements (MAJOR):
    Displays:1 display allowing an admin to insert major requirements for selected majors
    Navbar/Menu: Will direct user to other subsystems
    Buttons: 1 Logout button, 1 Submit button
    d. Faculty and Course Information (FCI):
    Not a feature in Admin view
    e. Course Grades (GRADE):
    Displays: 2 Major displays. One being a list of students. Another being a grade sheet of selected student which allows a teacher to change their grades,
    Navbar/Menu: Will direct user to other subsystems
    Buttons: 1 Logout button, 1 back button, one button for each student

---

f.) Informal design activities notes:

The goal was to design a simple interface that had the required fields for the users' log-in information, and then a screen that displayed each of the five subsystems for each type of user. The display of the main menu will look virtually the same for all users, with possibly differing color schemes in order to distinguish the screens of each type of user. Each menu will have a help function on the bottom left. If on a specific subsystem, there will also be an option to navigate to other subsystems through a menu at the top. Given that our current focus is back-end development, the current design of the user interface is still subject to change, especially for the interfaces of each subsystem and its functions.

g.) Design review notes (optional*)

Images of the SDSU campus were chosen as part of the background of the design and will be used for only the log-in page and the main menu page of the subsystems. We wanted to opt for a more simple interface, so we decided to choose a monochromatic color scheme for each page, with blue being the main color for the overall system. Moving forward, we opted for the implementation of a static navigation bar at the top of the screen that lists all the subsystems so that a user may switch between any subsystem at any given point. We decided that this would be a more efficient and user-friendly implementation compared to a button that directs a user back to the main homepage. So that the user knows which subsystem page they are on, the subsystem will be highlighted in a lighter color at the top navigation bar.

Section 8.)

a.) Code review notes (optional*): Our program code is primarily PHP and HTML with some CSS design elements. Our database was created with mySQL and SQL queries embedded in our PHP documents allowing us to access information within the database. Our program utilizes MAMP to run an Apache web server and execute our PHP files as web pages. The primary function of our web pages is to extract and display user data and perform other simple tasks that in general have a constant runtime. Our SQL queries access our relational database with the standard efficiency of a SQL select/update/insert statement without indexing.

b.) Unit test plan: The unit test plan will follow the normal method of unit testing where we test every unit independently as it is developed in order to ensure that the units are following their correct function. This method will save us time and effort from any future testing because it ensures that less mistakes and bugs will be made. We can do so by using an automated testing tool like selenium that helps make this testing process go more smoothly. Also, refreshing the website as changes are made to see if they have a positive or negative impact on the functionality/look and feel of the website. This would mean going through testing for each of the 7 different areas of the website we are working on. And testing each requirement in those areas individually.

Test case - Login functionality : test that each user stored in the database is able to log in. Also tests that every person is directed to the correct homepage.

Test case - Logout functionality: test that users are able to log out easily from any point.

Test case - Structure: test structure for each subsystem

Test case - Authorization: test that each type of user has the correct authorizations

Test case - Menus: test that menu functionality is working properly

Test case - Print reports: test printing out the records of each subsystem

Test case - Restrictions: test restrictions on certain users

Test case - Registration functionality: test that you can register for a class online and only one class at a time.

Test case - Search filter: test that the search filter for finding course can always find the correct class according to the set filters

Test case - Outline creation/access: test that only a major advisor or admin can create,edit, or look at outlines.

Test case - Modifying data: test that when data is modified through the website that the element that has been modified is correctly stored. Along with a tracked history of modifications made.

Test case - Change grades: test the functionality of teachers being able to change a students grade.

Test case - print grade report: confirm that students are able to access and print their own grades and nobody else's.

Test case - Queries: test that any queries made by a user return the correct information from the selected subsystem.

Test case - Help functions: Determines whether a help text-box pops up whenever a field is hovered over and ensures that each field has its own unique help output.

c.) Unit test review notes (optional*):

Originally, we had planned to execute many unit tests (as you can see above) throughout our project but we never really got around to it. Ideally, we would have used PHPUnit as our unit testing framework. In a black box sense, we had been unit testing just as much as we had been integration testing, but when it comes to a true whitebox unit testing, we have no tests to show.

d.) Unit test/retest results: N/A

e.) Integration test plan: We will continuously perform integration testing. During this testing we will run tests on many different functions that work together. Like registering for classes that you

find after searching for your major requirements. A test would simply find if you can register for a class through a method that is not directly related to the registration page. This testing will make sure that the data is processed correctly and all the different components of the application can work together without bugs. We can do so by using the website in different ways and combinations to see if there are any bugs between two unrelated methods. This can be done with our project by creating different combinations of the 7 different areas of functionality and their requirements. So that each subsystem will function properly with whatever framework of the signmeup system it is intended for and the general requirements of the website.

f.) Integration test review notes (optional*):

We have been very consistent when it comes to integration testing and blackbox testing as a whole. From our project's conception, we were consistently blackbox testing our code and web application. Originally, it was small things like testing that changes to our html code would be reflected on our website. Then we started database testing: making sure that changes and queries made from our website would be reflected in our DB. In database testing we tested single entry of information, ensuring things like user ID or specific course schedules do not already exist in the database. Additionally, we

Test case - General Requirements: test the general requirements

Test case - Framework: test the relationship between the structure/framework and subsystems of all the data

Test case - Electronic Student Record subsystem: test the functionality of the subsystem of student records

Test case - Course Registration subsystem: test the functionality of the subsystem of the courses and their current registration status.

Test case - Major Course Requirements subsystems: test the subsystem of the major requirements and that they correlate with the correct major.

Test case - Faculty and Course Information subsystem: test that the subsystem functions properly and has the correct faculty for the correct courses.

Test case - Course Grades subsystem: test the functionality of the grade subsystem and that grades change accordingly with the actions taken in the application.

Test case - Student User: Go through the process of what a student would do on the website and every path that they could take and see if the website adjusts/acts accordingly.

Test case - Admin User : Go through the process of what an admin would do on the website and every path that they could take and see if the website adjusts/acts accordingly.

Test case - Teacher User: Go through the process of what a teacher would do on the website and every path that they could take and see if the website adjusts/acts accordingly.

Test case - Data entry: Tests that data is to be entered only once and can be accessed from other subsystems data have been entered.

g.) Integration test/retest results:

As previously mentioned, we have been integration testing very frequently. During our testing processes, we have been able to ascertain certain requirements are fully functional and have been able to identify shortcomings and defects. Our first example of integration testing revolved around user login functionality. The user logs in with predefined credentials and depending on whether those credentials are in the student, teacher, or admin tables, our validation.php file would validate the entries and point the user to the appropriate home page. That was our first and most fundamental integration test. Many other integration tests resolved around inserting information into the database and having those changes reflected in other views. For example, when a student would enroll in a class, we would want those changes reflected in his student record under "Future Courses". The same thing would apply to the Drop Course functionality. There was a decent amount of trial and error between integrating two or more units. Sometimes functions wouldn't work at all, sometimes the functions would have defects and not display the correct information. But through our extensive testing, we were able to remove every defect we could find in our current prototype.

Section 9.)

Risk status / Areas needing further analysis / Questions still not answered / Plans for getting risks & issues resolved

When discussing risk status in terms of our project, some areas that need further analysis would include allowing our student to register for courses by clicking on them, and the fact that our group has different PHP versions, so that causes some of the functionality of our project to become unstable. These two risks are addressable, and plans to resolve them include creating some sort of link functionality for the courses, in order to allow a user to click on them. As for the PHP versions, in terms of our group, we all plan to switch to the version of PHP that allows us to be able to have consistency in the code of our project and to be able to work without issues.

With that said, these are two issues we have come across that are fixable in the near future, there are other questions that we have that we will investigate if there is ample time. One

such issue is the issue of password security. Currently, our database users use a 3 character passcode that consists of 3 integers. This would be very easy to guess and a good feature to implement would be a check to see if the password contains certain characters, digits and capitalization. Some other issues that are currently not high priority but could be investigated in the future include the following: a way to reset the password, a print report feature that could be implemented for various frameworks in our project, a no deletion feature, a feature that tracks approved major outlines for each student in each department, an outline access feature, a no deletion feature to protect said outline, a modification feature for said outline, and grade entry features, which allow grades to be updated or updated.

Section 10.)

a. Test Plan/Strategy
    i. Plan for testing project requirements
        1. Our plans for testing each requirement consist mainly of usage-based testing, functional testing, and graph based testing. The primary function of our program is to securely log data and respond to user requests. Our testing system will address each of the requirements by mimicking expected user actions. Our test plan will also utilize graph based testing to model the flow of information between user, user interface, and database. Database testing will be utilized to ensure data integrity is maintained. A detailed list of testing plans for each requirement will be provided in 10b.
        2. The tools used in our testing will include the database management tools phpMyAdmin as well as our local host web server and Visual Studio Code to manipulate and edit source code.
    ii. Informal notes related to test planning activities

    We decided to do a testing triangle between Team 4 (Team Mountaineers) and Team 1 (We Are the Ice Cream Man). We got together on December 6 in the library to test each other's project. It took about an hour and a half to two hours to fully test each others functionality. It was a good experience and felt it was necessary to have a third party with no hand in our project's development give pointers and have them run through the functionalities to detect any defects or shortcomings or any possible improvements to user experience.

    iii. Test plan review notes (optional*)

My group and I have opted for creating a test plan for functionalities we know we have implemented and are attempting to implement. To manage risks, we discarded several requirements like individual outline, and an admin's ability to add major requirements. So, because we have discarded them in our first prototype, we did not include them in our prototype's test plans. Eventually, when we finish all requirements, we will make a more detailed and expansive test plan.

b. Test Procedures/Cases
    i. For each test case
        1. Description of test case
        2. Requirements tested by each case
        3. Inputs for each test case
        4. Expected results
        5. Pass/fail criteria

Test Case ID	Test Case	Description	Steps to Take	Expected Results	Status (Pass/ Fail)
1	Login Improper Credentials	Login shall sent user back to	Input top field: test@gmail.com Input bottom field: 12345	Reload page/Send user back to default screen (login.html)	PASS
2	Admin Login	Test that certain credentials will navigate user to admin home	Input top field: nc@signmeup.edu Input bottom field: 293	Validation.php file will navigate user to adminHome.php	PASS
3	Student Login	Test that certain credentials will navigate user to student home	Input top field: g@gmail.com Input bottom field: 123	Validation.php file will navigate user to studentHome.php	PASS
4	Teacher Login	Test that certain credentials will navigate user to teacher home	Input top field: al@signmeup.edu Input bottom field: 190	Validation.php file will navigate user to teacherHome.php	PASS
5	Drop Class (Student)	Test that student can drop any registered courses	Navigate to Student Record tab and press Drop button under “Future Courses”	Unique course will be removed from display and success message will pop up	PASS
6	Consistent UI (Student,Teacher,Admin)	Test that all pages of student, admin, and teacher are the same	Login in to student and press each navbar item and check if UI is consistent. *Repeat for Admin and Teacher*	UI would be consistent, colors, fonts, boldness, overall style	PASS
7	Query Grades by Semester	Test that each dropdown option has appropriate	Navigate to Course Grades view and test that all drop	Spring 2022 will print specific courses from table,	PASS

	(Student)	values (Might need to refer to coursechedule and studentschedule tables)	down options reflect each semester when submitted	Repeat same thing with Fall 2022 and Spring 2023	
8	GPA & Credits (Student)	Test that GPA and Credits earned are correct	In Course Grades, confirm that GPA and credits are correct (An admin or teacher changing grades should should also change these fields)	GPA will change if an admin or teacher change grades and Credits should change if teacher or admin give a student an F	PASS
9	Register for Course (Student)	Tests that a student is capable for registering for a course	In Course Registration view, search by major and click 'Register' button for a course	New view will open up giving the user a "Registration Successful" message. New course should show up in Future Courses display in Student Record view	PASS
10	Single Entry of Information	Tests that certain information can only be entered once (EX: student should not be allowed to register for course that they are already registered for)	In Course Registration, try to enroll in something that already has the status of 'ENROLLED' from Student Record View	New view will open up giving a "Registration Failed" message	PASS
11	Query Major Requirements (Student)	Tests that querying major requirements pulls the correct information from the database	In Major Requirements view, choose each dropdown option, submit, and compare with major requirements table to ensure correctness	Should DIsplay appropriate list of classes	PASS

12	Query ALL Faculty, ALL Courses, ANY Semester (Student)	Tests that all courses from all teachers from a specific semester functions properly	In Faculty Course Information view, choose ‘All Faculty’ and ‘All’ courses in all 3 semesters (Refer to courseschedule table in db)	Should print a long list of Courses, professor teaching, and relevant course information like office location, office phone, date and time, etc	PASS
13	Query a single teacher ALL courses (Student)	Tests that all courses that a single teacher teaches prints properly	In Faculty Course Information view, Search Instructor: 12, Meredith Palmer, Search Courses: All, Spring 2022	Should print basic instructor information, as well as a list of courses with course number, course name, date and time, location, and total seats available	PASS
14	Query a single teacher, single course, single semester (Student)	Tests that you can query single teacher, single course, single semester prints accurate information	In Faculty Course Information view, Search Instructor: 12, Meredith Palmer, Search Courses: CS480 Operating Systems, Spring 2022. For no prerequisites Search Instructor 11, Axel, and Data structures for any semester	Should print Basic teacher information, specific course information, as well as prerequisites if there are none, under prerequisites tab will show ‘N/A’	PASS
15	Add User (Admin)	Tests that an Admin can add a user and give them student, teacher, or admin privelleges	In Home of Admin pages, Add a user by putting in a unique UserID and fill out other fields. (EX: 321, test@email.com, password, James, Bobby, 2345678989, 1992-06-06) Submit and see results	Should print ‘Failed to add user’ if data is missing or if any UserID is duplicated. Should print a success message otherwise	PASS
16	View Student	Tests that Teacher can	In Student Record view, click ‘View	Should display relevant information	PASS

	Record (Teacher)	view record of any existing student	Student Record' button for Caleb Greenfield	for Caleb	
17	View Student Record (Admin) (*Not Implemented*)	Tests that an admin can view and change record of any existing student	In Student Record view, click 'View Student Record' and Edit any of the fields	Should allow an Admin to change user data and have it reflected in the display	FAIL

18	Change Student Grade (Admin & Teacher)	Tests that both Admin and Teacher can change any student's grades	In Course Grades, click 'View Grades' for Caleb Greenfield and change any of the grades with dropdown feature, after changing, submit changes	Should print success message to screen and have it reflected in database and student record view for everyone (including that student)	PASS
19	Add Course (Admin)	Tests that an admin can add a course to database	Navigate to Course Registration and insert unique data for each field (EX: 321, CourseNew, CS450, 1, 0, 3)	Should print a success message to screen and have it reflected in courses table	PASS
20	Add Major Requirements (Admin) (*Not Implemented*)	Tests that an admin can add major requirements	Navigate to Major Requirements and use dropdowns to select which major and which class will now be required	Should print success message and be reflected in majorrequirements table in database	FAIL
21	Back Buttons	Tests that each 'Back' button will navigate user back to the previous page, discarding any attempted changes	Click each 'Back' button	Navigate user back to previous page and discard changes	PASS
22	Logout Buttons	Tests that logout functionality works	Click Logout button on any or all views	Should navigate user back to	PASS

Haley's Review Notes:

- 17: can view student record, cannot edit any fields
- 20: not implemented

- Very visually appealing, consist across pages
- Intuitive and user-friendly

Team evaluation by Group 6: Team TBD

Detection System	A: Place noise detection sensors within an area of 5 square miles	○
	B: Program the device to detect various types of animal noises	✓
	C: Request alert messages be sent to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise	✓
	D: Alert messages will contain the type of noise detected, the strength of the detected noise, and the location of the detected noise to within 3 meters.	✓
Controlling Computer	A: The controlling computer will be located in the park ranger station	✓
	B: It will save all mountain lion alerts received within the last 30 days.	✓
	C: It will save a summary of alert information for data older than 30 days, but received within one year.	✓
Controlling Program	A: Sound an alarm whenever an alert message is received from the animal detection system	✓
	B: The alarm will continue until the ranger turns it off.	✓
	C: Once the alarm is turned off, it will not sound again until another, separate noise is detected at a different location.	✓
	D: Allow the ranger to classify each alert as definite, suspected, or false, indicating the probability that a real mountain lion was detected.	✓
	A: A report showing all mountain lion detections by date detected and by classification (definite, suspected, or false)	✓
	B: A report showing all mountain lion detections at a specific sensor location.	✓
	C: A graphical report showing detections on a map of the park and areas within 2 miles of the park.	✓
	D: A report showing detection classifications by ranger.	✓
	The system should be developed in a way that would allow it to be easily reconfigured for other parks in the State of California.	✓

Team Mountaineers did a very great job for the scope of their project. All of their requirements were fulfilled. Some of the requirements were not but those were things like, placing a handful of sensors within a 5 mile radius. Instead of purchasing sensors, they opted for using their physical computer which was smart. Another requirement was detecting and classifying various sounds as a certain animal. This would involve some sort of AI or Machine Learning program which I believe is way out of scope of this class. Overall, this team did a very impressive job and met all requirements.

-Group 6 Team TBD (Reese Hartwell)

ii. Test procedure review/inspection notes (optional*)

After having another group test our project, they pretty much ascertained our original hypotheses. There was nothing truly unexpected when it came to the peer reviews. We did lack certain requirements in a few departments but we had previously acknowledged them and expected them to fail during the test. It was nice to have someone who had not developed our system to test requirements because it was an unbiased third party who had no part in its development, so they would ask questions if they had any related to design choices and whatnot. It seemed like our design choices were comprehensible and user-friendly which is great. Overall, there were no surprising shortcomings that we had not already addressed. They left promising feedback and stated shortcomings

c. Test Tool Development (if applicable)
   i. Test Tool Design: N/A
   ii. Test Tool Design review/inspection notes(optional*): N/A
   iii. Test Tool testing results: N/A
d. Test Results Report
   i. List of tests executed and results
      Refer to the table above.
   ii. Defect Discovery profile

      In terms of "defects", my group and I believed we had just one, and the test results confirmed that. The only known defect revolved around the responsibility of admin by creating a user and assigning them to student, teacher, or admin. Instead, an admin is just able to create a user but not assign them to a role. This is somewhat a defect because it's an important feature that prevents certain functional requirements, but we were aware of this shortcoming and had not implemented it yet. Another possible defect would be with Admin's functionality of adding courses. Ideally, we would want an admin to add a course and have students be able to register for that course in the next semester. However, we updated the course table of our database only instead of also updating the courseschedule table. Similar to adding a user and adding that user to student/teacher/admin tables, we had trouble adding a course and adding that course to another row in a table in one button press. Our tester, Hailey, had not explicitly identified these defects because they aren't necessarily intuitive unless someone had a hand in developing the application, but we are looking to complete them soon (probably after our deliverable is due).

   iii. Defect Closure Profile (Defect has been resolved/closed)

      Ultimately, we are still working on resolving our defects. We have not been able to consider a defect as "closed" yet but are working towards that goal of having no defects in our project. As this project would be a fantastic addition to each of our resumes, it would certainly be in our best interest to "close" these defects and complete

further requirements.

Section 11.)

a.) Timesheet for Team Members:
Timesheet Link