Senior Design Server/Client Development for Project Matching

Final Report

Team 44

Clients: Jacob Grundmeier, Akhilesh Tyagi

Advisor: Akhilesh Tyagi

Brady ~ Individual component design
Seth ~ Individual component design, Project Management
Kyle ~ Testing, Code reviewer, Git Documents manager
Matthew ~ Database manager
Yunhao Yang ~ Individual component design, Testing
Gavin ~ Full Stack Development
Jake ~ CICD manager
Tyler ~ Engineering Standards manager

sdmay22-44@iastate.edu

http://sdmay22-44.sd.ece.iastate.edu/

Executive Summary

Development Standards & Practices Used

Agile Practice

HyperText Transfer Protocol (HTTP)

ANSI/ANS 10.3-1995 Standard for Documentation of Computer Software);(ISO/IEC/IEEE 26514)

IEEE Code of Ethics

Summary of Requirements

Historically, the Senior Design Project Lifecycle has been a manual process requiring significant time, communication and organization from faculty. To significantly increase the efficiency of this process, a defined matching algorithm and structured web-server is needed for client-project proposition, student-project matching, team resource creation and project review scheduling; allowing for optimized team creation and communication.

Applicable Courses from Iowa State University Curriculum

COM S 311: Introduction to the Design and Analysis of Algorithms

COM S 309: Software Development Practices

COM S 319: Construction of User Interfaces

COM S 363: Introduction to Database Management Systems

New Skills/Knowledge acquired that was not taught in courses

Laravel, Angular, Typescript

Table of Contents

1 Team

1.1 Team Members

Kyle Kent

Matthew Karr

Tyler Staker

Brady Synstellien

Gavin George

Jake Gudenkauf

Seth Gardner

Yunhao Yang

1.2 Required Skill Sets for Your Project

Frontend knowledge, backend knowledge, database building, database connection, agile methodology, client interaction, CICD

1.3 Skill Sets Covered by the Team

Frontend knowledge, backend knowledge, database building, database connection, agile methodology, client interaction, CICD

1.4 Project Management Style Adopted by the Team

Agile methodology

1.5 Project Management Roles

Brady ~ individual component design
Seth ~ individual component design, Project Management (Boards, meetings)
Kyle ~ testing, code reviewer, Git Documents manager
Matthew ~ Database manager
Yunhao Yang ~ individual component design, testing
Gavin ~ Full Stack Development
Jake ~ CICD manager
Tyler ~ Engineering Standards manager

2 Introduction

2.1 Problem Statement

Historically, the Senior Design Project Lifecycle has been a manual process requiring significant time, communication and organization from faculty. To significantly increase the efficiency of this process, a defined matching algorithm and structured web-server is needed for client-project proposition, student-project matching, team resource creation and project review scheduling; allowing for optimized team creation and communication.

2.2 Requirements & Constraints

- Main Functional Requirements
    - 491
        - We need to use a thoroughly researched algorithm to match students to client projects
        - Faculty panel
            - We need to provide a scheduling system for Faculty reviewers and students to make the process more convenient for both
                - After each team is signed up for a panel, a form will be sent to faculty, and each faculty need to sign for a panel
        - We need to allow faculty to easily assign grades to groups using canvas integration
    - 492
        - Industry Review Panel
            - We need to provide a scheduling system for Industry reviewers and students to make the process more convenient for both
        - We need to allow industry reviewers to easily assign grades to groups using canvas integration
- UI Requirements
    - We needed perspectives for all of the following users
        - Students
            - Project Preference Form
        - Teams
            - Manage team files
            - Give access to design docs
            - Schedule review timeslots
        - Faculty Instructors
            - Approve projects
            - Add students to database

                - Send notifications to students
            - System Administrators
                - Manage non-student DB
                - Creation of team resources
            - Faculty Advisors
                - Assign grades to teams
            - Project Clients
                - Propose Projects
            - Faculty Panel
                - Schedule review timeslots
                - Input grades
            - Industry Review Panel
                - Access team design docs
                - Schedule review timeslots
                - Input grades
- Algorithm Requirements
    - Considerations
        - Market research (students)
            - Student polls
            - Possibly client polls
        - Client + faculty advisor
            - Tech skills sets needed by the client are represented
        - Abet Requirements
            - Team diversity
        - Objective function
        - Student Satisfaction function
        - Project Satisfaction function
        - Auction model
    - Constraints
        - Use of an SQL database:
            - MariaDB or MySQL
        - Use of a well documented web-framework
            - Laravel
        - Limited web server memory/resources.
            - Linux Virtual Machine
        - Security for user authentication:
            - Canvas sign in integration
            - OKTA or LDAP integration
    - Quantitative constraints
        - Limited user types.

        - # of students
        - # of Faculty Instructors
        - # Panel Reviewers
        - …
    - Limited development time
        - One semester of Planning
        - One semester of Dev
    - Limited ability for real time features
        - Time between requests and responses

2.3 Engineering Standards

- HyperText Transfer Protocol (HTTP)
    - Scheduling and Preference Requests
- File Transfer Protocol (FTP)
    - Upload initial student information/ design doc files
- (ANSI/ANS 10.3-1995 Standard for Documentation of Computer Software);(ISO/IEC/IEEE 26514)
    - We want to make sure our software is understandable and maintainable as we will be passing it on to a system administrator after development has finished

2.4 Intended Users and Uses

- The intended users of our project will be anyone involved in the process of assigning groups for Senior Design as well as anyone that is a stakeholder in a senior design group's project.
- This project benefits stakeholders as it speeds up the initial steps of team creations and early communication. It centralizes all data for the course and allows for better organization through the entire senior design lifecycle.
- Use cases:
    - Assigning groups
    - Assisting in group-client interaction
    - Making it easier for groups and stakeholders to communicate
    - Scheduling of faculty and industry panel review

3 Original Design

3.1 Design Context

3.1.1 User Needs

Faculty (Professor, Faculty Advisors, Administrators, etc…):

Professors need a way to approve viable projects, add students to a student database and notify students of important updates because they are responsible for doing these tasks. They are the most important role as they are vital for the flow of projects and students into the semester.

Faculty Advisors need a way to assign grades to groups because they are the final decider of team grades.

IT Administrators need a way to interact with the client and faculty database and retrieve team details because they are the role that will manage the client and faculty database and manually create team resources once teams are created.

Faculty Panel Members need a way to schedule panels, communicate with students and faculty because their role is important to review the design of teams at the end of the semester. They then need to communicate with the team they are going to review and the faculty advisors to help assign grades.

Clients (Project creators/proposers)

Clients will need a way to create and/or propose projects for 491. An example of a client user would be an iowa state professor, corporate sponsor of iowa state, student, and administrator. The defining characteristic of a client would be a user looking to develop a solution to a certain problem.

Students (Students Taking 491)

Students will need input on their project preferences because they want to be assigned into a project.

Outside ISU Organization (ABET/Industry)

ABET Evaluators need access to team documents because they need to be able to evaluate teams on their ABET values when the project is being finalized

Industry Panel Members need a way to access design docs and assign grades because they are the final evaluators of the projects and have a strong say on the project's grade after looking over the docs.

3.2 Design Exploration

3.2.1 Design Decisions

Design Decisions:

1. Laravel as backend framework
2. Angular as frontend framework
3. ISU provisioned server running on Ubuntu 20.04
4. Apache 2 as http server
5. MySQL as a database language.

3.2.2 Ideation

One design decision was what backend framework to use for the project. After an initial discussion with the IT admin on the project it was determined that an SQL based data structure was the preferred option for the backend side of the project. This was decided due to the availability of servers on campus and the lower cost associated with facility the backend on campus versus an alternative cloud option. Once SQL was determined we started to determine potential backend frameworks. The top backend framework options were Laravel, SpringBoot, and MariaDB. SpringBoot was one of our initially preferred options because most of us had prior experience with the framework, Laravel and MariaDB were preferred options because our IT admin for the project recommended the frameworks which worked well with prior 491 projects. Other alternatives that were thought about included ruby on rails and flask, these frameworks were included as options based on the widely available documentation and use in commercial application platforms. The decision was ultimately to go with Laravel because of an interest in learning further about this framework which was the top recommended database by the IT admin on the project.

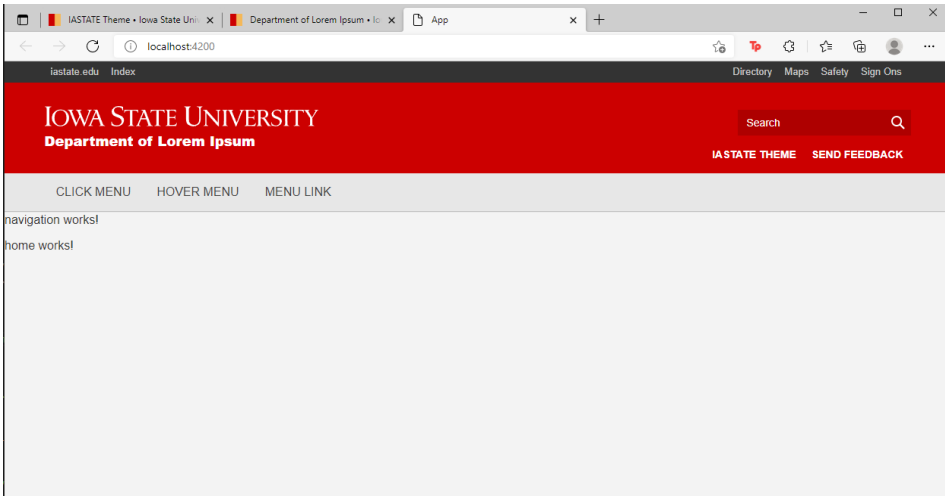
3.3 Proposed Design

We determined the front and back end frameworks, along with the initial designs. For frontend we selected angular as the framework and an initial outline of file structure. For the backend we selected Laravel and secured a server through ETG that can be virtually accessed.

3.3.1 Design Visual and Description

Frontend:

- For the frontend UI design, we used the Iowa State theme provided by the university, utilizing the generic nav bar given in the theme to make each view in our design as accessible as possible.



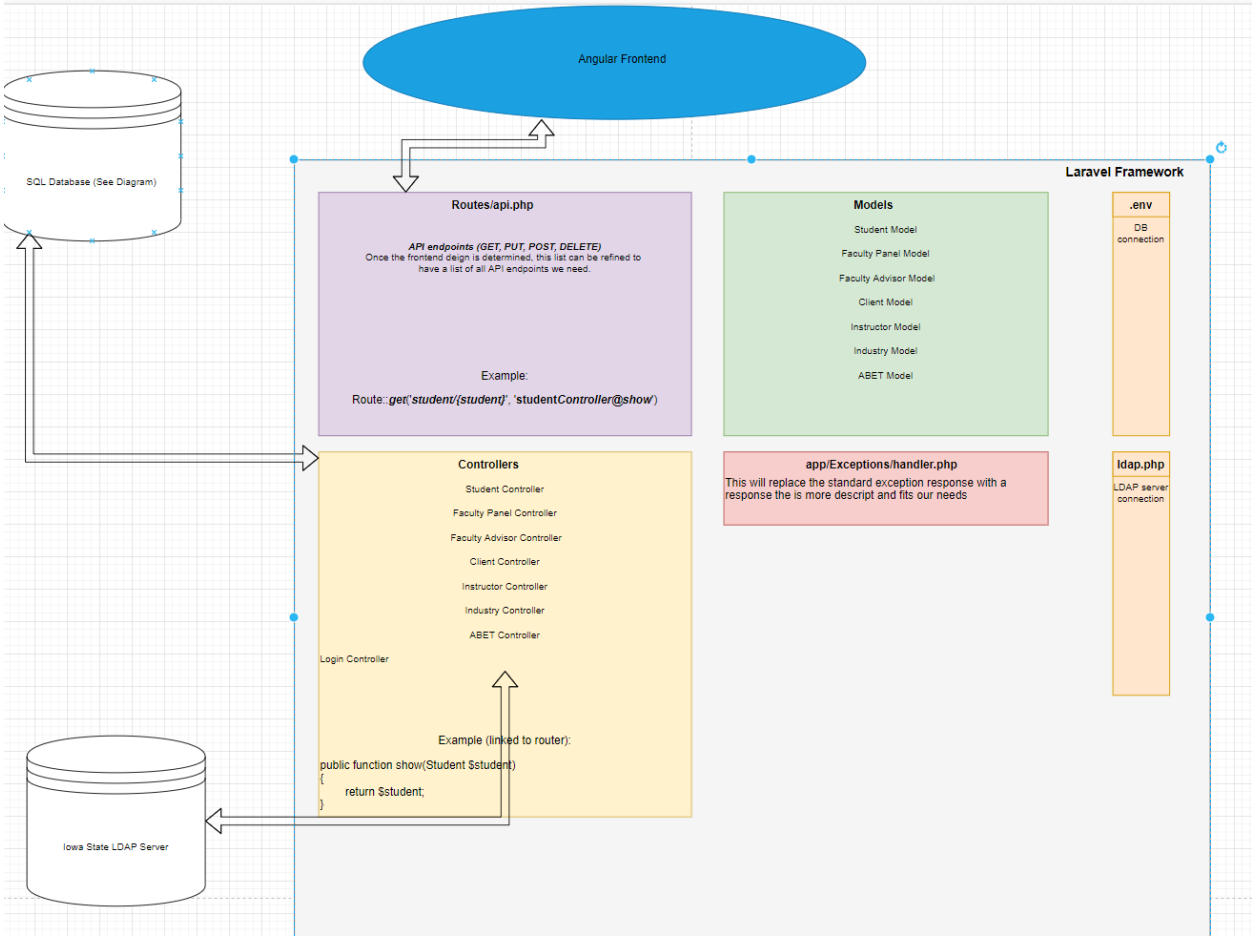
(Figure 3.3.1.1)

- For the frontend architecture, our initial design plan is to utilize angular components and services to create the 8 different views we need. To facilitate communication with the backend, we have resource services that manage all api calls and then have another set of services that handle the information received from or given to those resources. Our rough initial component diagram shows this concept with different architecture layers for the different responsibilities.

(Figure 3.3.1.2)

Backend:

- This implementation is very similar to any other Laravel application. There are going to be models for each user of the application, a controller to access the API endpoints for each user and routes to redirect the calls to the server to its respective controller. There is a connection to the MySQL database and LDAP server for user authentication. We also implement an exception module to handle exceptions so the frontend has a good idea of the errors. When an endpoint is hit, the router will send the request to the controller where, using the model, data will be queried from the database and send a response back to the frontend.



(Figure 3.3.1.3)

Database design:

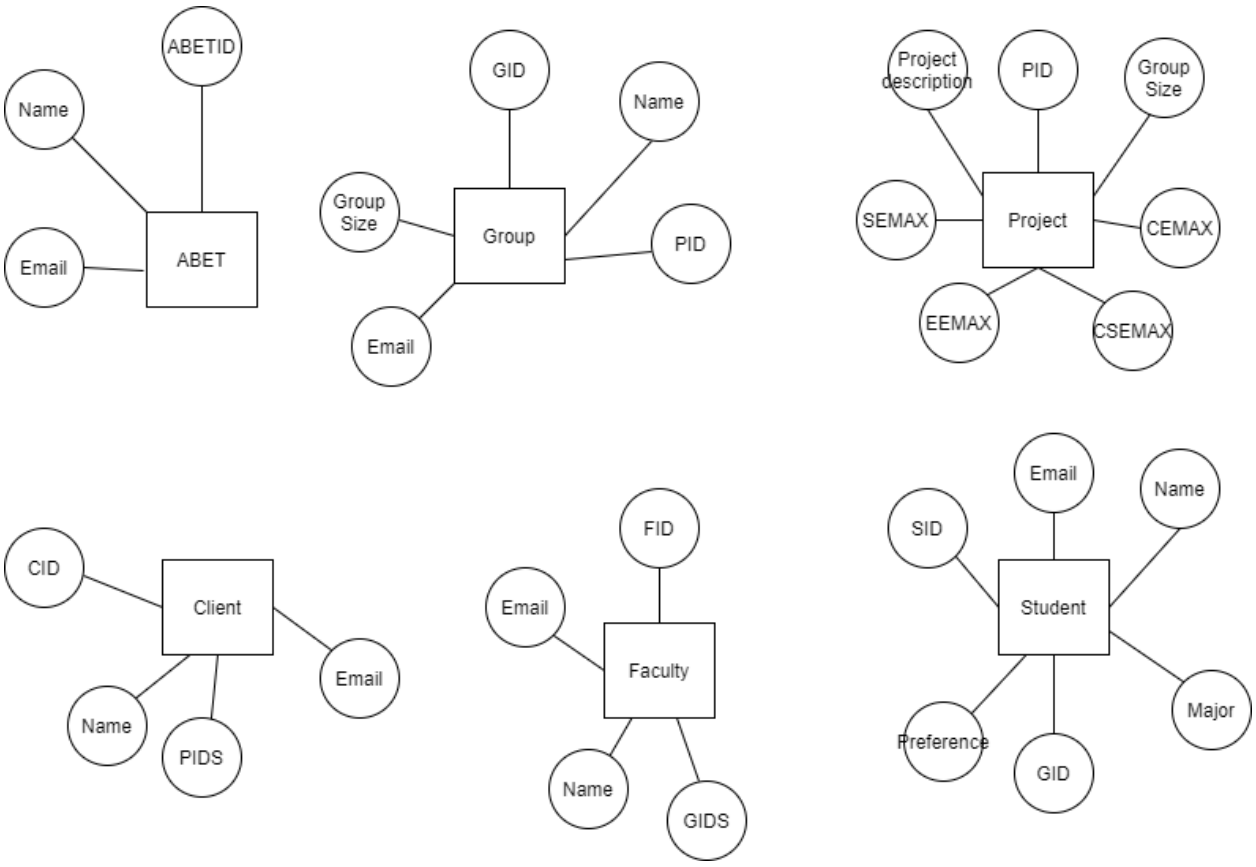
Fields that end with ID are markers for each table to have a unique identifier to call back to a certain line in the table.

Emails and names are included in all of the tables that have to do with people.

The fields ending in max have to do with the max amount of a major can be slotted into the project.

There is a PIDS field for client because clients can propose multiple projects.

There is a GIDS field for faculty because a faculty member could be assigned multiple groups or no groups.



(Figure 3.3.1.4)

Algorithm

Matching Overview:

For each student, check each project teammate and skills, and give a score based on the matched skills, has preferred teammate and is preferred project

If a new student is added in the project, the algorithm will do a rescan for all student to check if the added students the preferred teammate of them

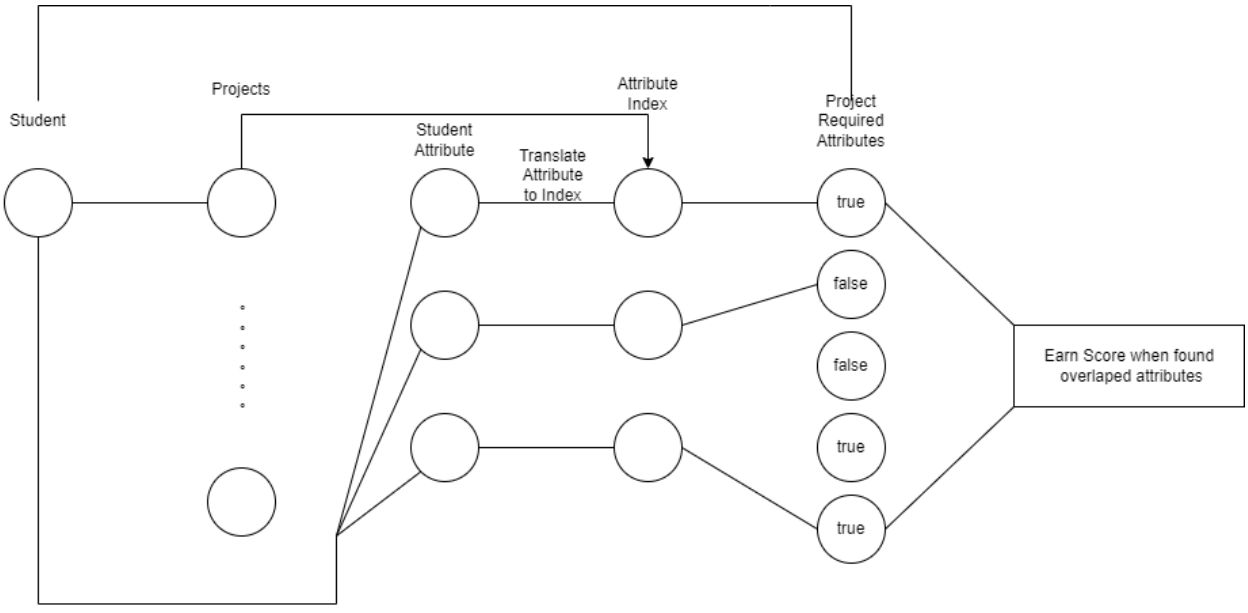
If the student found the project is the preferred project or has the preferred teammate, the algorithm will add the bonus score to that project for that student

Main Matching Data Structure

Project Attribute - A boolean array which contains all of the attributes appeared in the current semester projects. If the value is true, it means the attribute is required by the project

Student Attribute - A string array which contain all of the attributes of the student

Attribute Dictionary - A dictionary that can translate attribute name to attribute index



(Figure 3.3.1.5)

3.3.2 Functionality

Our initial design clearly shows how we will handle users' roles across our website. How a user will flow between the pages, depending on what roles they have. Additionally, on the backend side of things, we have defined what backend framework we are going to use. Last, we have defined our initial tables that need to be created in our database. Overall, our initial design documents provided our team a great starting foundation for our project.

Our initial design showcases what the first iteration of our web app will look like. If you click on the navigation bar you will be redirected to the new page instantaneously.

3.3.3 Areas of Concern and Development

One of our initial concerns was handling authentication with users, especially those users who do not have an ISU account/email. As a group, we came up with a solution to create our own dual authentication service. It may be a bit more complicated, but it is the only solution we have to our problem.

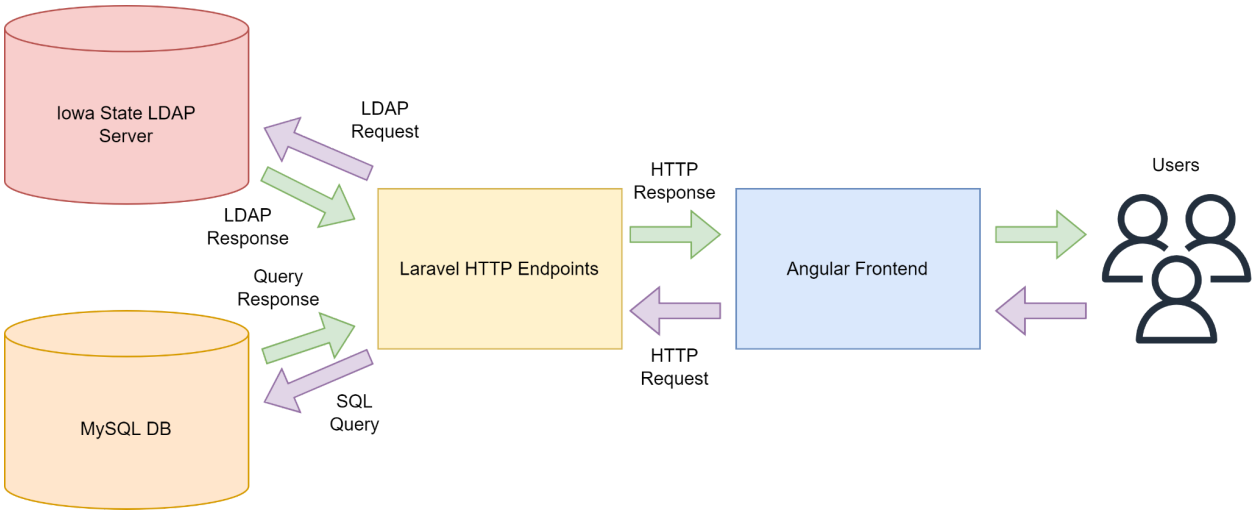
Another problem that we thought of comes into play with our algorithm. In order to satisfy an ABET requirement of diversity, we need to have diverse groups. However, one problem is that we do not want to discriminate against people because of their race, gender, or ethnic background. As a group, we have sought guidance from our client on how to handle this situation.

3.4 Design Analysis

Using all of the above sections to take into consideration, we have created our final project. We compared different frameworks to use on both the frontend and backend. Along with which type of database would be best for our client, tech stack, and knowledge from the team. Additionally, we took into consideration, which would be best in a tech stack and how the stack would run on the Ubuntu VM. Throughout the analysis process we met as a team along with our client and faculty advisor to help us decide which path was the best to go down.

3.5 Design Plan

Our design plan is based off of the high level diagram seen below. We modeled what our system should look like after choosing the technologies our project would use to meet the user's needs. We planned on designing a Laravel application that was able to connect to an ISU authentication server and a database on a VM. We then planned on designing a DB that was able to handle the volume of data we used and designed a MySQL database and the required tables in siad DB to meet data requirements. Next, to interface with the Laravel backend that was serving data, we planned to design an Angular application that was capable of linking the users to the application. Lastly, unconstrained from the design of the web-application, the algorithm was planned to be designed with the capability to read the data from the DB and be able to solve the problem of matching teams; meeting the weighted function requirements.



4 Changes To Our Original Design

4.1 Database

The database has changed to reflect the changes proposed by the backend and frontend. This meant adding a table for student skills and requirements. Adding all the foreign key dependencies. We also added database seeding for testing the database and filling the database with generic information.

4.2 Algorithm

The algorithm is changed from per project and gives each student a score to per student and gives each project a score. This change can better handle the new student added to the project team and handle the student replacement for each project team. Also, the algorithm no longer checks if both students want each other to be preferred teammates. It will add a bonus score as long as it finds preferred teammates in a project team. This change can simplify the score calculation process while maintaining the accuracy of scoring. The last change of the algorithm is an automatic replacement has been added to the algorithm. If it finds a team is full but a new student wants to join, it will check each teammate's current score and if possible, replace a teammate that has a lower score than the new student. This change can better improve the accuracy of the matching algorithm.

5 Testing

Some unique testing challenges that our design entails are:

- Algorithm analysis with different variations of filled out responses to a form for assigning 491 projects.
- Actions/accessible features for different user types on the front facing application site.
- Unique post data bodies for routes set up on the backend of the application.
- Error handling in the event of invalid data during interaction on the front facing application site or trying to receive response from the backend.

5.1 Unit Testing

- Algorithm
    - Enter multiple different groups of data to ensure the algorithm is working as predicted.
    - Input different sets of the data (different student data, and different group requirements) to test the project ranking number generator to ensure the algorithm generates a proper rank number based on the input data.
    - While matching with teammate, make sure the algorithm can find the most proper project based on two students' preferences
- Database
    - Write tests for all of the stored procedures created for the MySQL database. This would be written with the MySQL language inside the database. Schema testing check to make sure all the tables show up in the schema.
- Frontend
    - Each Angular component and service has its own test file, where we unit test methods using Jasmine and Jest.
- Backend
    - Laravel uses the unit testing framework called unitPHP. With this we can test all aspects of the backend functionality. We would test the controller, service, and other various internal backend components.

5.2 Interface Testing

- Frontend
    - Within our component test files we can test the Angular forms we create to ensure that data is properly passed into our service and resource calls to the backend.
- Database
    - Stress testing to make sure that when information is added to the database it's not lost nor slows down the database. Testing with Neo4j and postman.

5.3 Integration Testing

- Frontend
    - Puppeteer for e2e testing and capturing performance to analyze.
- Backend
    - The testing framework that we choose for the backend is Pest. This will help to ensure that the entirety of the backend produces the correct behavior.

5.4 System Testing

Ensure that the integration and unit tests that test the key features of the system requested by the client are successful.

Stress test system to make sure it can handle an expected number of users within our time requirements.

Run the algorithm on an expected set of data and verify that it runs in a timely fashion(we have no constraints) and that it produces a client accepted output.

5.5 Regression Testing

By using automated test runners like Jest we can easily ensure that previously written tests still pass and the behavior of methods and components haven't changed.

Some critical features that we need to ensure do not break are the algorithm, the UI input for the algorithm, and extended parameters for the algorithm as these are key features requested by the client.

5.6 Acceptance Testing

- Frontend
    - Before we push any live changes to the site, we would want to have a quick meeting with our client or send screenshots of the design. This way if our client wants to change any visual or feature they would have a say before we push the change.
- Backend
    - Group members on the backend sub team, would meet and make sure that the new code changes have the correct functionality. Basically just have everyone on this sub team sign off on the changes.
- Algorithm
    - Explain the entire algorithm to the client step by step and show the code in the meantime.

5.7 Security Testing

- Research some common attack patterns and verify that our system can handle and prevent them from working by returning error codes or handling the attack properly.
- Make sure for our frontend components, that if a person is not signed into our site they should not have access to these views.
- Make sure for our backend controller links (requests), that if a person is not signed into our site they should not have access to these requests.

5.8 Results

Overall, our project has four main components. Each component has its own testing criteria that it must pass. Looking at the table below, you can see a summary of each component. Each component has their own set of requirements that they must pass to ensure that our project is completely tested.

Components	Summary narrative
Frontend	Users will be able to navigate the site freely with whatever role(s) that they have. Should pass all unit and integration testing. Lastly, should be accepted by the client before any changes go live (big features and UI).
Backend	Only internal users have access to requests. Passes all unit and integration tests.
Database	All data is returned correctly and precisely. Data return for a query does not take more than a second. Data added to the database is added correctly.
Algorithm	Algorithm will be able to generate a most compatible result base on each project's requirements and student's preferences

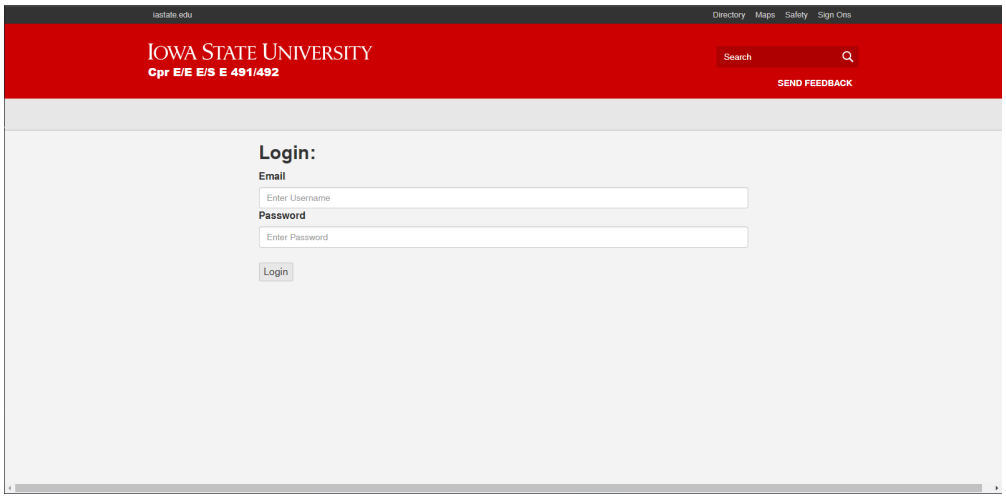
6 Implementation

https://git.ece.iastate.edu/sd/sdmay22-44

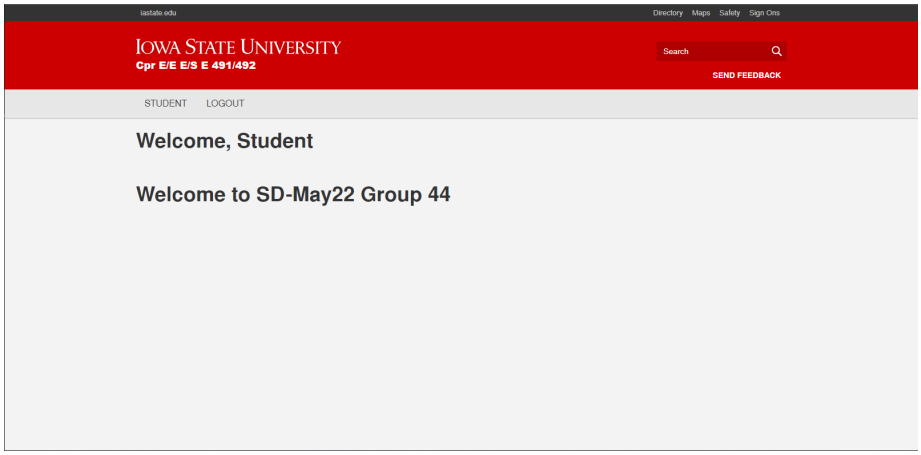
6.1 Frontend Design

Shown below are various views that exist within the application.

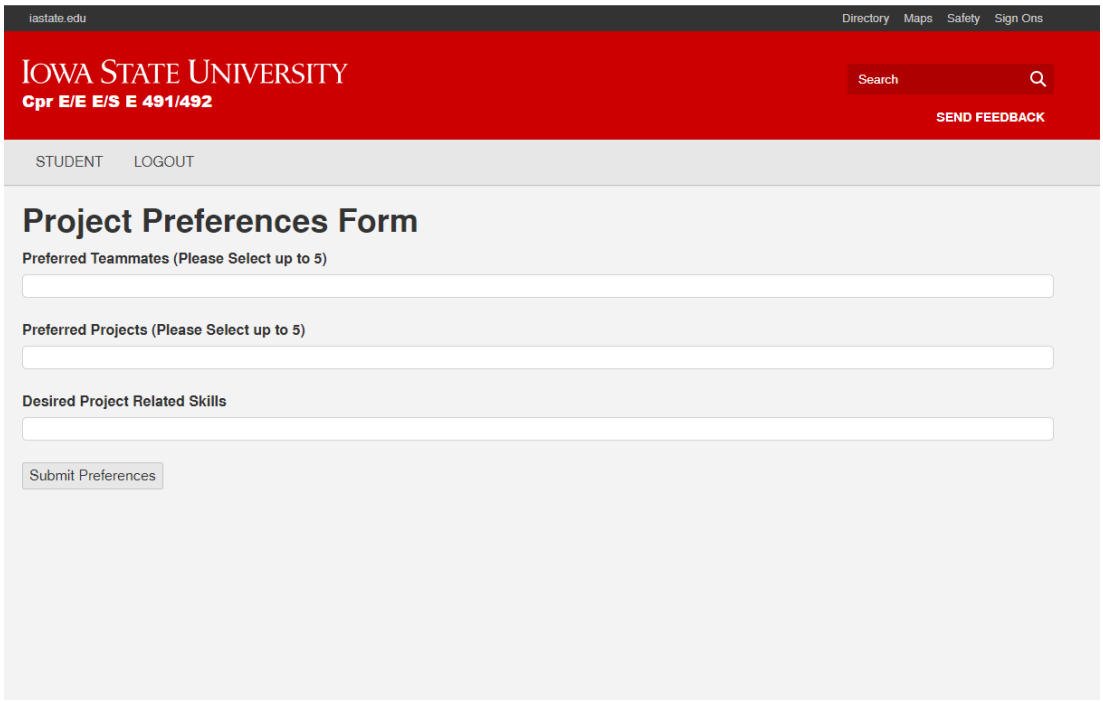
View that everyone sees when prompted to login.



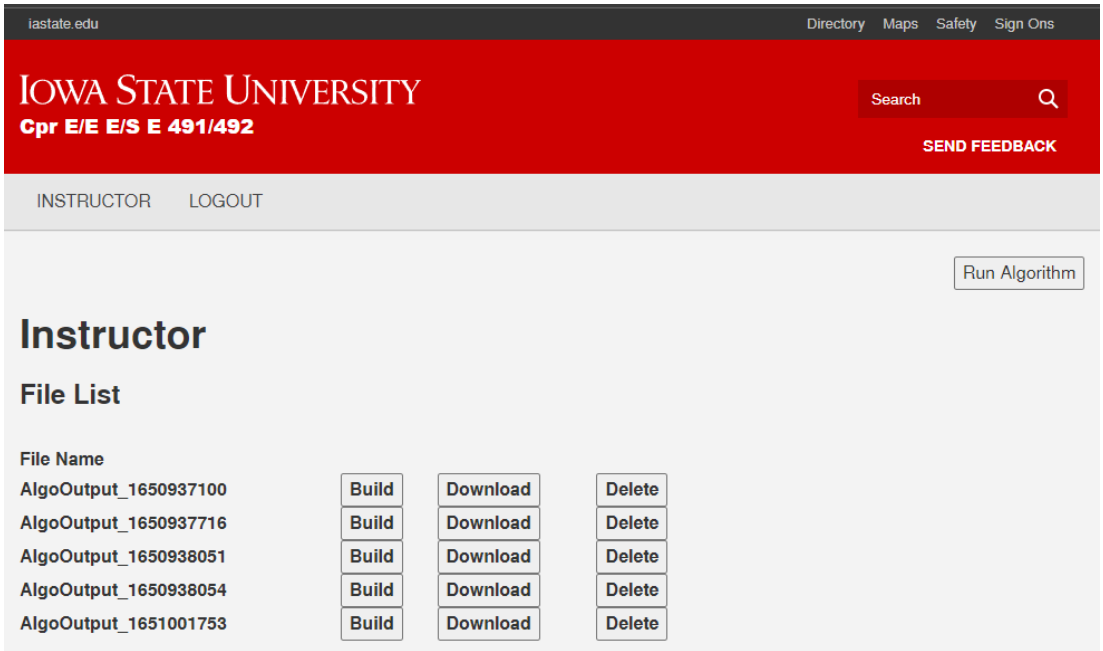
When a student is logged in they are shown the student tab and options appear for them once clicked.



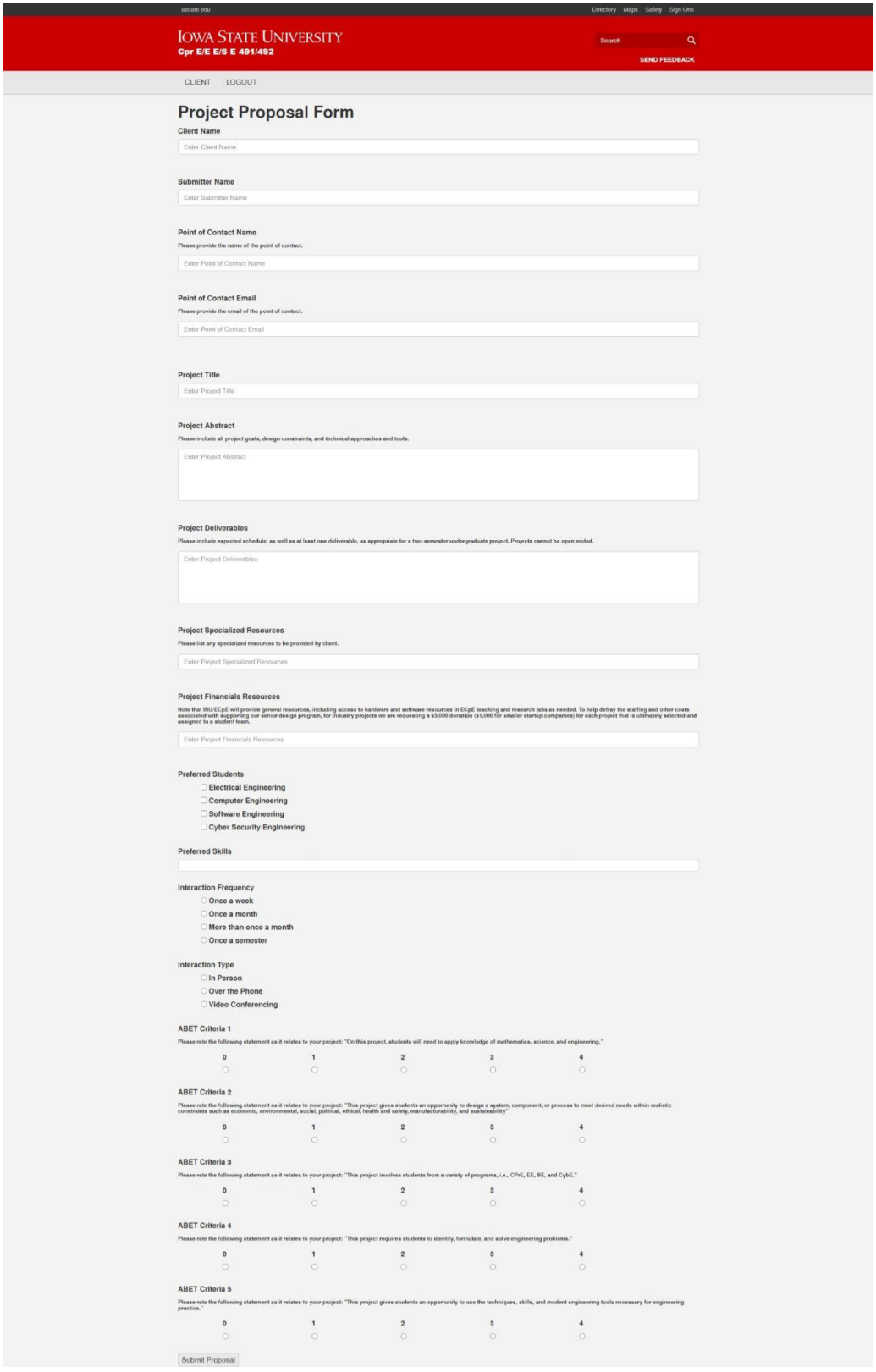
Below is the form students are presented with at the beginning of the semester. Students can select up to five different preferred teammates, preferred projects, and desired skills required for a project.



Shown below is the instructor view. Here, instructors can run the algorithm multiple times. After each run, they are able to download the file generated. After selecting the desired matching set, they can directly build the file to the database.

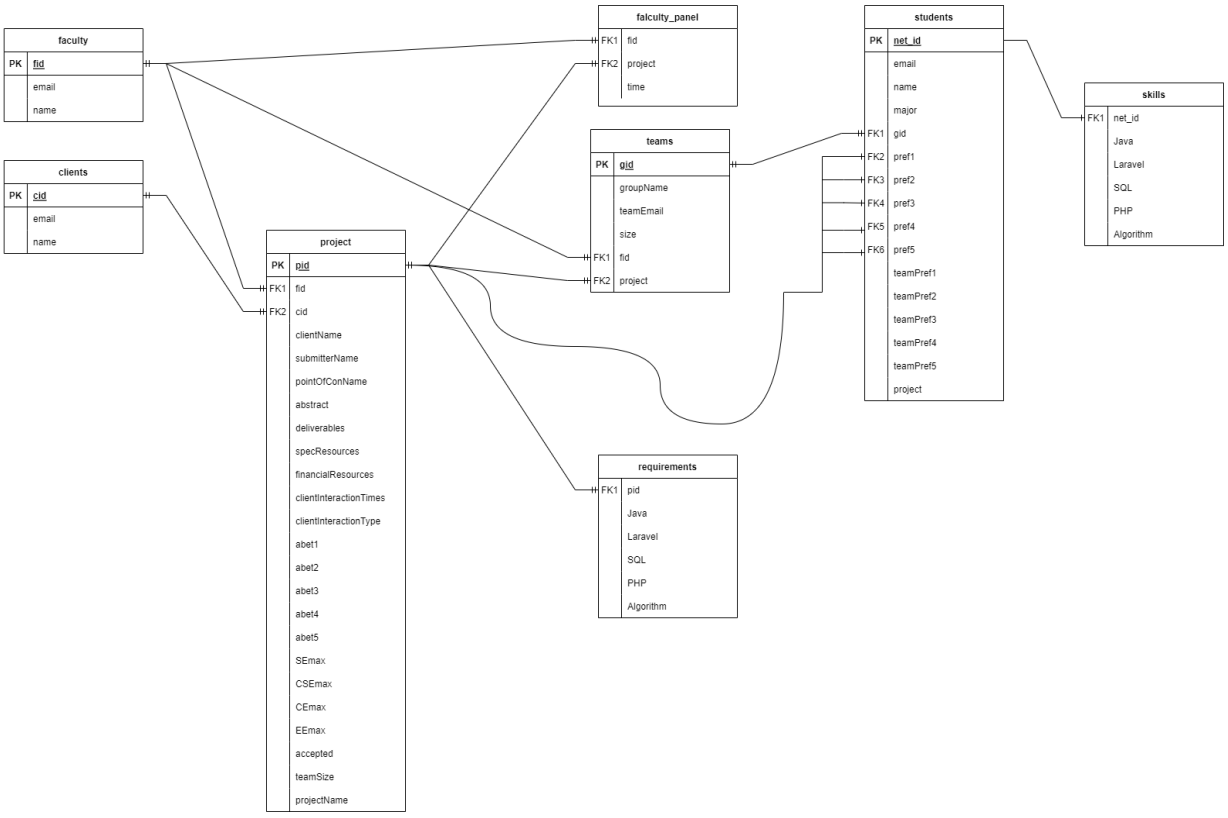


When a client would like to propose a project they are brought to this screen. Here they fill out all information related to the project and will submit it.

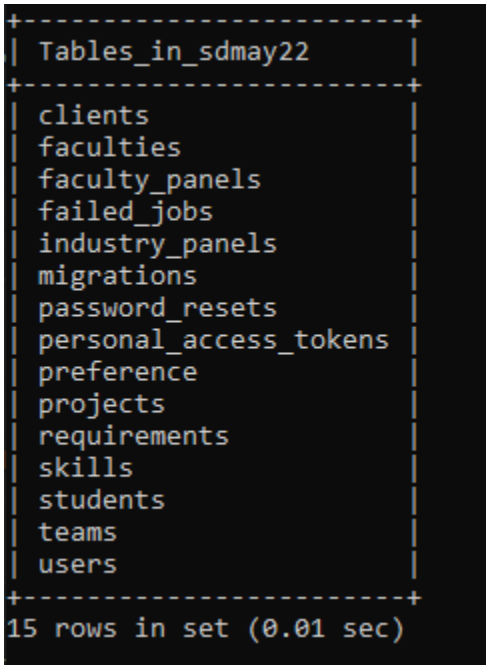


6.2 Database Design

Below shows how all the different tables are related to each other.



Total Tables that exist within our database.



```
+-------------------------+
| Tables_in_sdmay22       |
+-------------------------+
| clients                 |
| faculties               |
| faculty_panels          |
| failed_jobs             |
| industry_panels         |
| migrations              |
| password_resets         |
| personal_access_tokens  |
| preference              |
| projects                |
| requirements            |
| skills                  |
| students                |
| teams                   |
| users                   |
+-------------------------+
15 rows in set (0.01 sec)
```

Seeded data to populate our database, which allowed for better testing of the system.

project	pref1	pref2	pref3	pref4	pref5	teamPref1	teamPref2	teamPref3	teamPref4	teamPref5	gid	net_id
0	33	4	36	26	45	2CdyQMiCfq21JcA	0MeI6F4dFSSHXWL	null	null	null	0	0LzqiMPlKXIu4VD
0	34	40	44	42	12	vmvS4h6z70HOAjQ	0twCutzQPHPRBb2	null	null	null	0	0MeI6F4dFSSHXWL
0	43	34	43	41	33	WYSZsfBAcnckeGi	OBA8uJiFPi2VJz1	null	null	null	0	0mMPxypfRit96Hw
0	6	13	41	36	15	V7O2ur98UKLQTpZ	TpYPj3SDljymkoI	null	null	null	0	0twCutzQPHPRBb2
0	44	45	21	26	31	VGIFrBAyHGbJyBn	TpYPj3SDljymkoI	null	null	null	0	0vTcnfEoHoEvgAK
0	15	22	39	28	16	9jGU6z3xSanKFuP	OBA8uJiFPi2VJz1	null	null	null	0	1BvvWU2n7w6gdcx
0	17	24	19	18	22	IJz3Lwsavyav3dA	rv7euHoh76AUq2I	null	null	null	0	1HngOIQn8k945wH
0	12	10	45	22	38	q23mBNiuEQocLUh	58cFuSk6T4hW8kK	null	null	null	0	1NPqWD6xb0Ujs5
0	18	4	2	36	18	pqtFumSGDvwBUtY	aFNgxfYZ8docCo2	null	null	null	0	1qTvI0XEuvo33v4
0	16	28	7	39	4	MG0iLVmJJzGE44G	UVZrjTAzYObWJIK	null	null	null	0	1SlN483JWKT5Qyq
0	39	49	28	40	2	2tQ7XycNrPztNeH	kdYrIqQ6lOrCJAT	null	null	null	0	1uMdfTb6IkqP9wj
0	15	6	14	14	12	XzQerSamfHkXYks	AkB35ScQdOGoOW9	null	null	null	0	1uQ5qoUhKXfLNyn
0	32	2	15	46	24	TzjT3cIaHgW8jr4	UyFxHnL1QrZAd1x	null	null	null	0	1ZccpEqruCHSgiJ
0	31	25	16	29	18	YyU9nek6juifkZ1	57qjvMXTVycRP2b	null	null	null	0	2CdyQMiCfq21JcA
0	29	11	22	21	17	KdnjNdF2pMaCVyX	2CdyQMiCfq21JcA	null	null	null	0	2tQ7XycNrPztNeH
0	22	47	37	15	36	HoMzEYGcdRQcryl	scnPsHxOMMnvdEp	null	null	null	0	2WP7D2X9mZ6i3AA
0	41	11	16	7	47	wDCbXhakgmW7DSP	oYJiNJEt1P0pFuB	null	null	null	0	31Q1Wb7v9YLkyxv
0	34	24	27	8	49	V5kjA6ilUmEWsoI	K4QqPYJI6EZTagk	null	null	null	0	3cjkhleEMSOiujE
0	30	43	8	49	23	zOOSGbHpB3fF5yB	CFuI8iEEX2j1VKk	null	null	null	0	3F06tqKze6v6X6d
0	1	36	13	4	27	BXa1EHzVKTqeVfc	ASnSxbZEUl6dH1i	null	null	null	0	3FABCSz6TtRRQw2
0	7	20	45	10	34	Pz3hrbnzwXIQvyu	w24CFdV59CFJ0Hj	null	null	null	0	3mtgxkuhAl6Iplf
0	8	46	34	15	24	ar4S93zIF7Sw6Yu	jOPIRgorKdWWCVw	null	null	null	0	44vKkMfq1HCwaao
0	45	31	11	29	11	0LzqiMPlKXIu4VD	9UQ00WxxZVoQ4dQ	null	null	null	0	4h5rq9MXyNPw4Kb
0	15	8	35	20	6	6o9BHqjD1qgS1bM	vGaZQkeQUXYRT0p	null	null	null	0	4JN23M38rFAwOmv
0	26	43	20	49	34	rjD6zL57gMNz4L0	jh73gADCjHmpT0b	null	null	null	0	4sObafufXMjdsEn
0	18	39	45	30	41	57qjvMXTVycRP2b	7k9W4f51eqHLP2A	null	null	null	0	4W6oIKLKJL8YkcV
0	46	15	30	13	44	WC7iTlqTRBHE2js	V5kjA6ilUmEWsoI	null	null	null	0	4ZQtTjBLW76r9FY
0	34	16	34	46	7	ar4S93zIF7Sw6Yu	1NPqWD6xb0Ujs5	null	null	null	0	51XlT7sCLHT2rM4
0	27	41	30	11	49	Zb9ZEmlIa4oepny	WovBDUXoaq7ih0P	null	null	null	0	55jchkotU6nNYjP
0	49	24	6	2	31	YfvcTqI1kUhxV02	qtCszAR1ErqMO7b	null	null	null	0	57qjvMXTVycRP2b
0	36	32	44	10	16	3F06tqKze6v6X6d	TpYPj3SDljymkoI	null	null	null	0	58cFuSk6T4hW8kK
0	2	6	2	28	26	pWWGYmQTfTC3Nkw	FbHl7VRwLi6dL0A	null	null	null	0	5NZOU7pUhpyFyov
0	5	8	38	42	14	vlQ89nrEURPCItt	OtH59igZdanm42F	null	null	null	0	5OBOXZJ5Wa15rPm
0	32	38	34	46	29	p9x1Ye5hezhwyjc	JqSl311HvCSJFmV	null	null	null	0	5wxqf7Vyub0fa55
0	19	13	34	15	40	aZLyujfcGbGkZK7	PIknfHamEaPWIt3	null	null	null	0	61IkdFoQ0nKZ20Z
0	46	48	21	21	36	8uAQsJpsXAAtMgh	Bbn7Mx6utjKxL9e	null	null	null	0	6DipEpg6iSIr6VL
0	10	3	10	33	34	XzQerSamfHkXYks	i9hVt3Nnv9LpXzF	null	null	null	0	6o9BHqjD1qgS1bM

6.3 Algorithm Results

After running our algorithm on randomly generated data, the following matches occured.

```
300 assigned
Number of people have at least 1 matched skills: 259 out of 300
Number of people in their preferred project: 240 out of 300
Number of people working with their preferred teammate 140 out of 300

Number of people both have skill and in preferred project: 203 out of 300
Number of people both in preferred project and work with preferred teammate: 140 out of 300
Number of people both work with preferred teammate and have matched skill: 106 out of 300
Number of people have all attributes: 106 out of 300
```

7 Closing Material

7.1 Discussion

Throughout this course we worked together as a team to deliver this product to our client. We learned everything related to the full stack application implementation process. Along with a web application, we implemented our very own algorithm that matches students to projects. From starting up the server locally, to getting everything merged together and running on the server provided to us. Problems seemed to come up in many different parts of the application, but we solved them together in order to complete this project.

7.2 Conclusion

We released a working version of the product to our server. As a team, we divided our responsibilities between the three different areas of the project: frontend development, backend development, and algorithm design. Working in parallel aided in completing the project by our deadline and presentation to industry members. We believe that the product that we produced, was done to the best of our ability, given the time frame. Overall, we all enjoyed working on this project and interacting with our client.

7.3 References

No References

7.4 Context to Other Products

Other similar products that exist in the market but are not a copy of ours. A similar product that we discovered is Canvas Infrastructure. Each course is similar to our application in the sense that there are students, teams can be created and course faculty can create projects and assign students to them. It appears that web-apps that are course specific are less common and are hidden from the general public.

7.5 Appendices

Appendix 1: User manual

System Requirements:

This application is a web-app that will require a machine that is capable of running a web server. This can be done through any application such as Apache2. A backend requirement is the installation of PHP and Composer. Node package manager is a frontend requirement.

Setting up the database:

For setting up the database we will want to first initialize a database. This can be done either through a linux virtual machine or by initializing it on your home machine. We will need the host, port number, user, password, and the name of the database. We will use these fields to place inside of the env file which can be found in the backend\sdmay22.

```
DB_CONNECTION=mysql
DB_HOST=
DB_PORT=
DB_DATABASE=
DB_USERNAME=
DB_PASSWORD=
```

Once these are plugged into the env file the laravel migrations will be able to access the MySQL database specified. Before we can run any of the migrations you will need Composer. Once Composer is installed we can now begin running the migrations. You will want to get to the directory which contains your env file. Then you will want to run php artisan key:generate. We need this for running the artisan commands. To run the migrations we use php artisan migrate. This will build the tables in the specified database. A helpful command if you decide to tweak the migrations is php artisan migrate:reset. This command will drop all of the tables in the specified database. For generating test data we can use the command php artisan db:seed this will run the seeders which will fill the database with data.

Changing the database and adding seeders:

Editing the tables in the database can be accomplished by going to backend\sdmay22\database\migrations. This folder is where the migration code is placed. Once the files have been edited we can run php artisan migrate:reset and then php artisan migrate. This will update the database with the changes to the tables. The migration files are run alphabetically so to maintain foreign keys the tables that are used in foreign constraints must come first alphabetically. For adding seeders we place the files inside backend\sdmay22\database\seeders. We can then run php artisan db:seed to run the seeder files.

Modifying the Frontend Code:

Making any changes to the Angular code in the project will require a rebuild of the Angular project. This build contains all modules combined into a file that will be referenced by the Laravel application when deployed. The build can be created by using the custom command “npm run build:prod” while in the Frontend/app directory. This will generate and copy the build files to the necessary directories in the backend. It is important to note that for the build to compile successfully, you will need to have the npm package manager installed on the machine and have all the dependencies for our project installed using “npm install” while in the Frontend/app directory. This will generate the node modules folder that will contain these dependencies, and is necessary for compilation.

Alternatively if you want a local instance of the frontend that will update live with changes, you can simply use the Angular “ng serve” which will start the frontend on the specified port and will use the custom backend IP address that you may specify in the environment.ts file.

Setting Up Laravel:

This application uses multiple Composer packages, so to begin by running *composer install* to install the packages. Next, you will need to configure the .env file. Filling out DB connections and LDAP secrets are the only requirements. Lastly you will need to run *php artisan jwt:secret* to initialize JWT for auth. You will want to run *php artisan migrate* to create database tables.

Modifying the Backend Code:

To modify backend code, the desired changes must be made and file saved. It would be important to push code to a source code control application but it is not required. The web-app will automatically update any changes to laravel code while the app is being served.

Modifying the Algorithm:

There are 5 main global parameters in the algorithm.

The first parameter is ProjectBonus. This parameter controls how many scores would be added to the project if it is a preferred project of the student. The parameter is an array, and the 5 values in the array means the scores to add when it finds it is the first to fifth project on the list. Adding more values to the array can allow students to choose more than 5 preferred projects

The second parameter is SATURATE. This parameter decides the rate of score added when it finds duplicate attributes. It currently only works on skills

The third parameter is SkillBonus. This parameter decide how many score to add when it find a matched skill

The fourth parameter is TeammateBonus. This parameter decide how many score to add when it find there is a preferred teammate in the project

The fifth parameter is RECURSSION_CAP. This parameter decides the levels of recursion of the optimization function. This parameter is mainly designed for efficiency concerns. The algorithm will be slower if having at too large RECURSSION_CAP

Appendix 2: Alternative or Initial designs

At first we planned on using okta for verifying users into our website but we just ended up using ldap for the verification. We also wanted to verify users outside of ISU. There was no real good way to add outside users so we modified this functionality to use a custom authentication for non-ISU users. The non-ISU users have to be created by an admin.

Appendix 3: Other Considerations

Our dependency on the third party LDAP for the authentication caused us to have some panic when our credentials expired near the end of the semester. While trying to wrap up testing on the

endpoints, we suddenly encountered an invalid credentials error coming from our authentication on the backend.