Senior Design Server/Client Development for Project Matching [Phase 3]

Final Report
Team 02

Client-Advisors	Dr. Akhilesh Tyagi Jacob Grundmeier
Database Design	Noah Nelson
Frontend Design	Evan Brummer Noah Nelson Robert Holeman
Backend Design	Evan Brummer Max Kueller Noah Nelson
Algorithm Design	Robert Holeman Devin Tigges

Email	sdmay24-02@iastate.edu
Website	sdmay24-02.sd.ece.iastate.edu

Table Of Contents

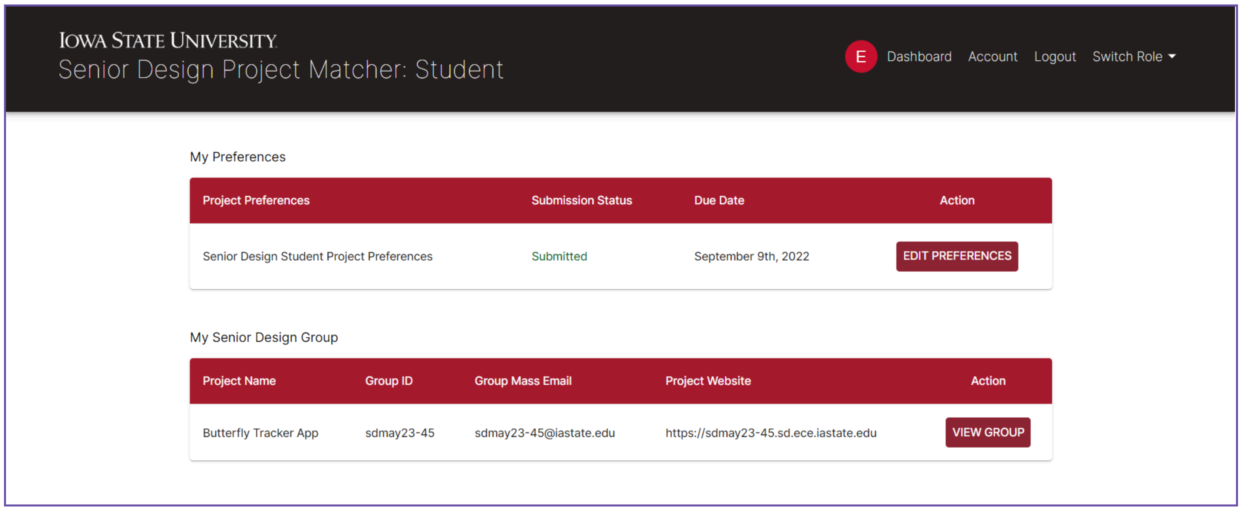
Introduction

Problem Statement

Every fall/spring semester, the Electrical and Computer Engineering (ECpE) department at Iowa State University assists faculty in approving and assigning design projects to students enrolled in the ECpE senior design program. This includes Electrical, Computer, Software, and Cybersecurity Engineering majors. These design projects are submitted as proposals by clients both in and outside the university.

The current process for managing senior design projects requires the use of multiple third party tools and many hours of manual organization. Potential clients are prompted to fill out a PDF form that must be processed manually after submission. Upon approval, the proposal is made visible to students (i.e. posted on Canvas) and added to a Google Form that students use to indicate their project preferences. All of these preferences are collected in a spreadsheet until the deadline, at which point the course faculty do their best to sort through student preferences and assign them each to the highest possible project on their list.

In the interest of saving time and providing a consistent, reliable method to collect, manage and assign senior design projects, our team has been tasked with the third phase of creating an all-in-one web application for the ECpE department. The site will be used to simplify client proposal submissions, assign teams of students to projects using a robust matching algorithm, and automate many of the other project management steps that faculty/instructors had to tend to in previous Senior Design semesters.



My Preferences

Project Preferences	Submission Status	Due Date	Action
Senior Design Student Project Preferences	Submitted	September 9th, 2022	EDIT PREFERENCES

My Senior Design Group

Project Name	Group ID	Group Mass Email	Project Website	Action
Butterfly Tracker App	sdmay23-45	sdmay23-45@iastate.edu	https://sdmay23-45.sd.ece.iastate.edu	VIEW GROUP

Revised Design

Functional Requirements

- The application must support 5 types of users that will use the app throughout a SD semester: Instructor, Advisor, Student, Client, and Board (members).
- The application must be able to intelligently parse CSV lists of user info (email, name, type), uploaded by Instructors to register users.
- Instructors must be able to manage rosters and user info at any scale, from large CSV imports to manually adding/updating/deleting attributes of individual users.
- All users are required to login with an email that has been added/whitelisted by an Instructor and assigned at least one role (user type).
- The login page must accommodate both logging in via the Iowa State Single-Sign-On (SSO) and with a standard email and password, because some Clients may not have a registered NetID.
- Users must be able to have multiple roles (user types) assigned, with the option to switch between them and view the corresponding dashboard.
- Users (typically instructors) must be able to open multiple sessions on multiple user types for convenience when setting up for the semester.
- After being registered, Clients must be able login, create proposal forms and submit them for review by an Instructor.
- Proposals should contain the client/company/organization name, Client name and email, project title, short description (project abstract), expected deliverables, specialized resources they can provide, anticipated cost, financial resources they can provide, preferred majors and quantity/percentage of each, other desired skills, anticipated client meeting/interaction frequency and format, and a brief questionnaire to indicate the nature of the project.
- Instructors must be able to approve and reject project proposals.
- Students must be able to declare their top [5] project preferences from the list of approved projects, and update their preferences at any point before the matching deadline.
- Instructors must be able to configure and send automated emails from inside the application. This is ideal for prompting clients to submit proposals, prompting students to enter project preferences, and other general notifications.
- Instructors must be able to execute a project matching algorithm to automatically assign teams of students to approved projects based on their preferences.
- Instructors must be able to modify the results of the matching algorithm to account for special or unexpected cases.
- The matching algorithm should generally find an outcome that assigns students to their most preferred project(s) as much as possible.
- Instructors must be able to assign Advisors to one or more generated project groups.
- Students, Clients, and Advisors of created project groups must be able to view their group info. This includes the project, group ID (e.g. “sdmay24-02”), group email, group SD

website, Client/Advisor names and contact info, and Student group member names, majors, and contact info.
- Board members must be able to sign up for one or more “industry review panel” timeslots near the end of the 492 semester.

Non-Functional Requirements

- All site features and functions should be intuitive and easily visible for all users.
- The site should be extensible, using well organized code and best practices.
- It should be easy to reset/re-use the application between design years.

Engineering Standards

- **React 18.2.0** functional programming and component design.
- HTTPS standards and status codes from the Internet Engineering Task Force **RFC 9110**.
- **HTTPS/CORS** fetch protocols from the Web Hypertext Application Technology Working Group (WHATWG).
- Standard practices for **representational state transfer (REST)**.
- Oracle **code conventions for Java** development.
- Best practices for developing with the **Spring Boot** Java framework.
- Rules for **relational data models** and database integrity.
- Basic branch, issue tracking, and merging standards for **Git**.
- **Agile** software development practices.

Security Concerns & Countermeasures

Protecting Data In Transit

HTTPS encrypts all of our client-server interactions to keep confidential user data and project data safe from network eavesdropping.

Role-Based Access Control

To protect confidential information, both the React app and Spring Boot server restrict user’s data access according to user roles and login credentials.

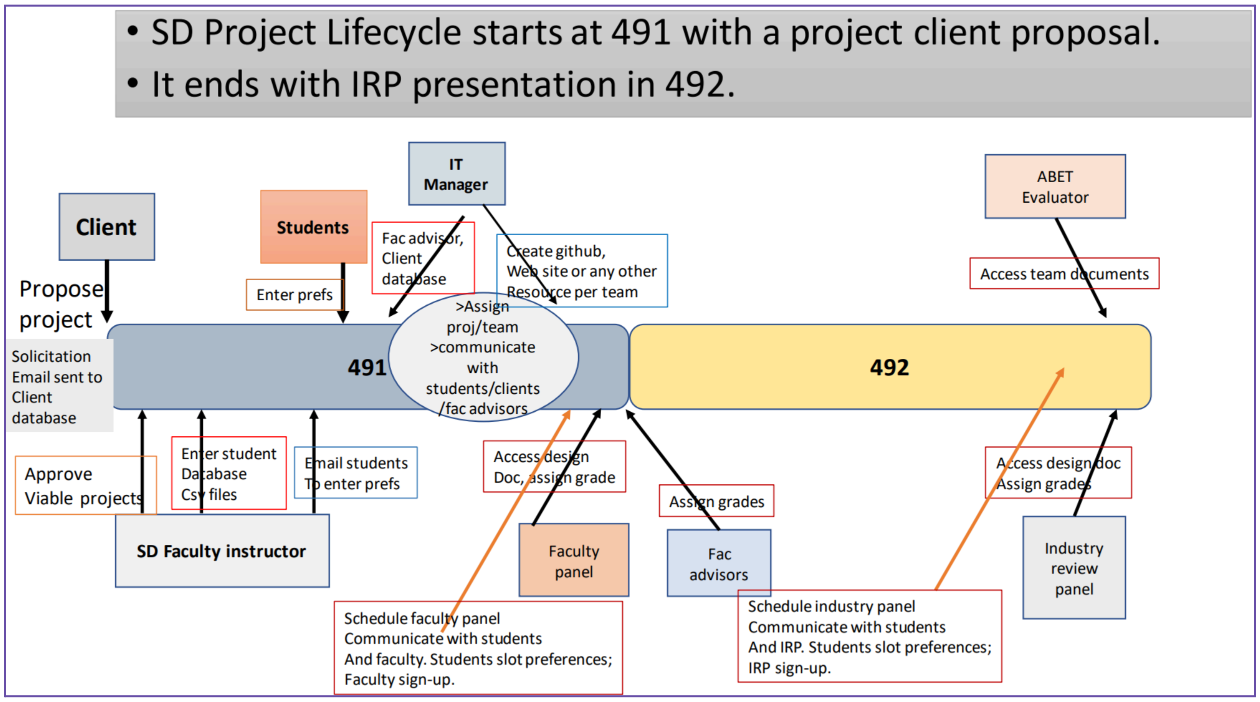
Preventing Injection

Our Spring Boot server uses Spring Data JPA abstractions to separate user input from core database transactions, preventing malicious query/code injection and allowing us to avoid writing raw SQL.

Evolution of Our Design

Frontend Feature Completeness

Our main priority throughout the design semester was learning our way around Phase 2’s React app and comparing the UI/navigation to the original requirements and lifecycle diagram.


• SD Project Lifecycle starts at 491 with a project client proposal.
• It ends with IRP presentation in 492.
Client
Propose project
Solicitation Email sent to Client database
Students
Enter prefs
IT Manager
Fac advisor, Client database
Create github, Web site or any other Resource per team
>Assign proj/team >communicate with students/clients /fac advisors
491
492
ABET Evaluator
Access team documents
Approve Viable projects
Enter student Database Csv files
Email students To enter prefs
SD Faculty instructor
Access design Doc, assign grade
Faculty panel
Assign grades
Fac advisors
Access design doc Assign grades
Industry review panel
Schedule faculty panel Communicate with students And faculty. Students slot preferences; Faculty sign-up.
Schedule industry panel Communicate with students And IRP. Students slot preferences; IRP sign-up.

Senior Design Lifecycle Management System. Provided by Dr. Tyagi.

When looking through the source code of the app for the first time, we located several areas for improvement. Each page had lots of hard-coded placeholder values and example content (likely for demo purposes). Throughout our design semester, we made small incremental changes for issues we found early on. As we started work on the algorithm and server endpoints, new requirements occasionally arose for the frontend, leading to new iterations throughout the implementation semester.

Choosing the Backend Framework

The original plan from our client-advisors was to build the backend server with PHP Laravel. While this aligned with the toolsets they were familiar with, they realized that needing to learn these new tools during the implementation semester was slowing our team down. They eventually allowed us to build it using the Spring Boot Java framework instead.

Rebuilding the Algorithm

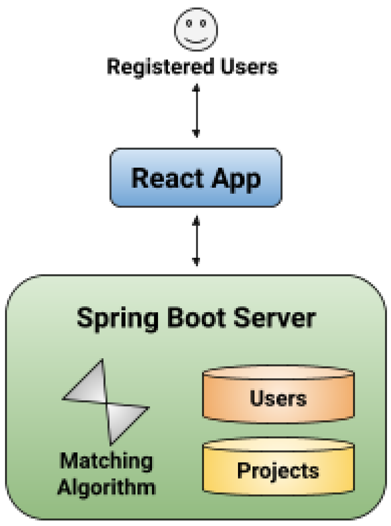
Based on the work of Phase 2, we put together a new points-based bidding algorithm that takes several user and project factors into consideration. This new design accommodated the evolving requirements of our implementation cycle.

Implementation

Design Details

Overview

Our design follows a layered architecture approach. It consists of a ReactJS application that is deployed alongside a Spring Boot web server. We used Jakarta JPA to manage the relational databases for users and projects. The matching algorithm is built into the source code of our Spring Boot application.



React Frontend

The React application uses Routes for the overall structure. For data persistence between components (and between refreshes), we implemented a UserProvider context to save, retrieve and share the data from the browser's session storage.

There are some generic Routes that are accessible to all users, and some that require a user to have a specific role to access (for example, Students are the only users that need to access /preferences to edit their project preferences). Built-in routing and navigation tools allow us to require users to complete certain steps before visiting other parts of the app (e.g. forcing all users to log in first).

The file structure is broken up into reusable components, CSS content, images, boilerplate for testing, and the user pages at each Route path.

```
                                 src/
           └components                      └user pages
             └HelpButton.js                   └board member
             └NavBar.js                       └client
             └ScreenLoader.js                 └faculty advisor
               ...                            └instructor
           └css                               └student
             └App.css
             └index.css                       └Account.js
               ...                            └Dashboard.js
           └images                            └Login.js
           └tests                              ...

                                            └App.js
                                            └UserContext.js
                                            ...
```
The file/directory structure of the React frontend.

Spring Boot Backend

The backend application is split up into modules for the algorithm, CORS configurations, constant values for user info, endpoint controllers, data transfer objects (DTOs), relational entity data structures, custom exceptions, database repositories, and service classes.

The algorithm uses data structures for students, projects, and preferences, and can create a set of project assignments when called from the AlgorithmController. User records can be buffered and stored with the CsvController. There are additional controllers for Projects, Proposals, Teams, and Users.

DTOs for login, projects, proposals and users are used when handling HTTP requests in the controllers. These are converted to their corresponding relational entities via the user service.

```java
@PostMapping("/create")
public ResponseEntity<UserEntity> createUser(@RequestBody UserDTO request) {
    try {
        UserEntity response = userService.createNewUser(request);
        return new ResponseEntity<>(response, new HttpHeaders(), HttpStatus.CREATED);
    } catch (UserAlreadyExistsException e) {
        return new ResponseEntity<>(new HttpHeaders(), HttpStatus.CONFLICT);
    } catch (Exception e) {
        return new ResponseEntity<>(new HttpHeaders(), HttpStatus.BAD_REQUEST);
    }
}
```

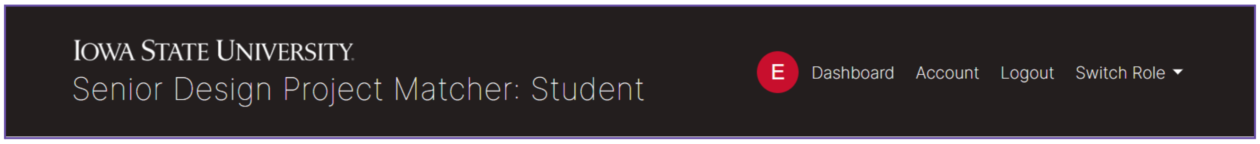
A post mapping from /controller/UserController.java.

Functionality

React Frontend

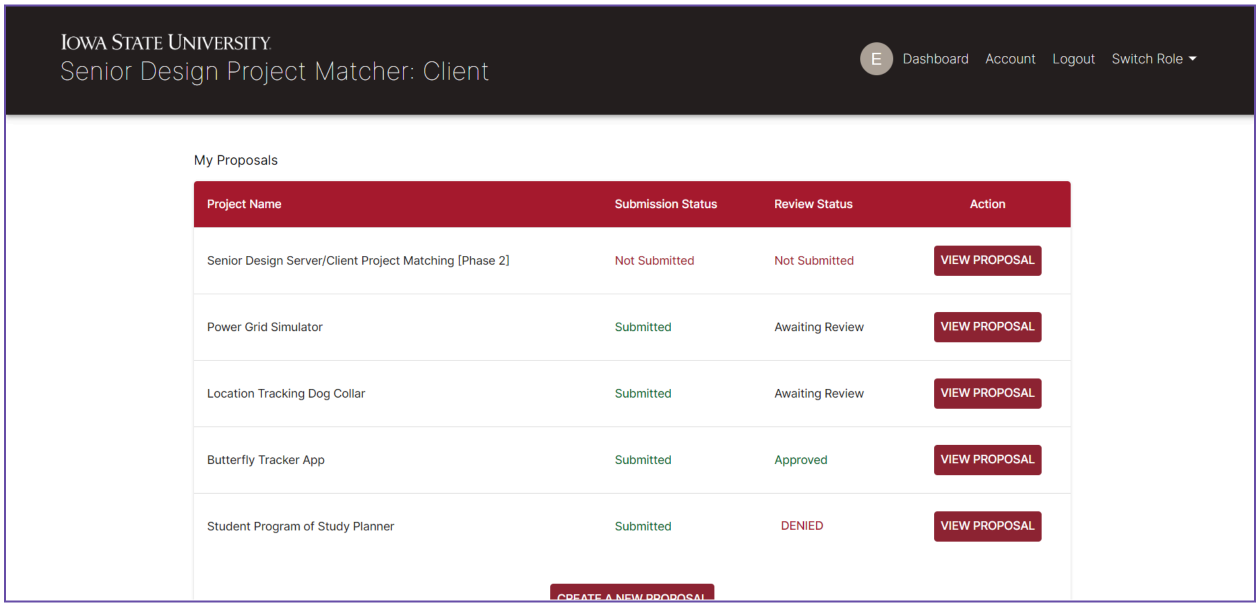
The frontend is powered by a React app that leverages the MUI library to create a cohesive and engaging user experience. Our app utilizes a multi-page routing system to aid in user navigation and create a directed experience. Information and data retrieval is powered by the React Asynchronous Fetch Library to communicate with our backend API endpoints.

Each page uses a ThemeProvider constant for the app's red/dark gray color scheme. Every page includes the NavBar component at the top, which provides menu options to return to the Dashboard page, view Account information, Logout, or Switch Roles (if the user has more than one).



The NavBar as it appears in the React app.

Page content varies slightly in implementation (different team members working on different pages), but they follow a general format of information tables and action buttons.



My Proposals

Project Name	Submission Status	Review Status	Action
Senior Design Server/Client Project Matching [Phase 2]	Not Submitted	Not Submitted	VIEW PROPOSAL
Power Grid Simulator	Submitted	Awaiting Review	VIEW PROPOSAL
Location Tracking Dog Collar	Submitted	Awaiting Review	VIEW PROPOSAL
Butterfly Tracker App	Submitted	Approved	VIEW PROPOSAL
Student Program of Study Planner	Submitted	DENIED	VIEW PROPOSAL

CREATE A NEW PROPOSAL

An example of the table structure with action buttons in the Client dashboard.

Communications with the backend are initiated with fetch() requests which contain extensive logic for safely handling responses and displaying results to the user.

```
fetch('/team/delete-all', {
    method: 'DELETE',
    headers: {
        'Content-Type': 'application/json',
    },
})
.then(response => {
    if (!response.ok) {
        throw new Error('Failed to delete teams');
    }
    console.log("Teams deleted successfully");
    // Once teams are deleted successfully, make a request to run the algorithm
    console.log("Starting Project Matching ... ");
    return fetch('/algorithm/run/1', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
    });
})
.then // ...
```

A fetch request triggered in the Instructor Matching dashboard.

Spring Boot Backend
We constructed a Spring Boot service for our backend. Through our endpoints, the frontend is able to access the data for students and projects. Further endpoints perform the other CRUD operations including adding and removing students and approval projects.

All of the general server functions are triggered by HTTP requests to the endpoints.

The application initializes JPA repositories to persist info in relational data formats.

```java
@Repository
public interface UserRepository extends JpaRepository<UserEntity, Integer> {

    boolean existsCurrentUserByUsername(String username);
}
```

The UserRepository interface declaration using JpaRepository.

The CommandLineRunner includes additional logic to automatically seed the databases using Faker. This is ideal for developing and testing with large amounts of data in mind.

Database
Our database is a structured SQL database. Accesses to our database are handled by the JPA/Jakarta. The database contains tables for Users and Projects, and may make use of Team and Proposal repositories in the future. These databases are relational.

Algorithm
Our algorithm is based on the early design of the previous team. Students submit bids on various projects or preferred teammates. This allows them to not only choose their preferences but also weigh the importance of their preferences. The goal of the algorithm is to maximize the number of students that get their bids.

```
MatchStudentsToProjects(students, projects):
  for each student in students:
    highestProjectBid = getHighestProjectBid(student)
    highestGroupmateBid = getHighestGroupmateBid(student.getPreferences())

    if highestProjectBid is null and highestGroupmateBid is not null:
      highestProjectBid = getFirstValidProjectWithGroupmate(student.getMajor(), highestGroupmateBid.getMajor())

    if student.getProject() is not null:
      sum = getGroupmateBid(student.getID(), student.getPreferences()) + getProjectBid(student.getProject().getID(), student.getPreferences())
      if sum > getProjectBid(highestProjectBid.getID(), student.getPreferences()):
        highestProjectBid = student.getProject()

    student.setProject(highestProjectBid)

    while highestGroupmateBid is not null and highestProjectBid is not null and not projectContainsMajor(highestGroupmateBid.getMajor(),
          highestProjectBid):
      setStudentPreferenceNull(student.getID(), highestGroupmateBid)
      highestGroupmateBid = getHighestGroupmateBid(student.getPreferences())

    while highestGroupmateBid is not null and highestGroupmateBid.getProject() is not null:
      total = getGroupmateBidSum(student, highestGroupmateBid)
      if total < getGroupmateBid(highestGroupmateBid.getID(), student.getPreferences()) and
                      projectContainsMajor(highestGroupmateBid.getMajor(), highestProjectBid):
        students[highestGroupmateBid.getID() - 1].setProject(highestProjectBid)
        break
      else:
        setStudentPreferenceNull(student.getID(), highestGroupmateBid)
        highestGroupmateBid = getHighestGroupmateBid(student.getPreferences())

    if highestGroupmateBid is not null and highestProjectBid is not null:
      students[highestGroupmateBid.getID() - 1].setProject(highestProjectBid)

    while highestProjectBid is not null and getNumAssignedProject(highestProjectBid) > highestProjectBid.getNumStudents():
      worstStudent = getWorstStudent(highestProjectBid)
      worstStudent.setProject(null)

  for each student in students:
    if student.getProject() is null:
      anyOpenProject = getAnyOpenProject()
      student.setProject(anyOpenProject)

  return students
```

Pseudocode for the main matching algorithm function.

Implementation Notes

In the initial project proposal, our client was intending to create a Laravel project. After some discussion, we decided to move forward with a Spring Boot project instead. This decision fit better with our skillset, the courses taught at Iowa State, and the industry norms. Similar logic went into the decision to use React instead of Angular in previous phases.

Testing

Processes

We employed various testing methods to ensure that our app both functioned as intended and met the requirements of our users. After running the tests we discussed the results and decided our next steps.

System Testing

In order to develop a proper plan for the semester, our team needed to understand and validate each component that was given to us by the previous team. In our analysis we found that they left a lot of the backend and front-end data handling for us to complete.

Functional Testing

Our team functionally completed dashboards for the most critical roles in our application including student, instructor, and client. In order to prove the functionality of our system, we demonstrated the correct function of each piece of our app against our requirements.

Smoke Testing

We utilized smoke testing frequently to test integration between our various teams and dashboards. One great example of smoke testing came from our successful Student-Client integration. Early on our team identified the essential functionalities of the dashboards including submitting student preference forms and project matching. Proving these functionalities early on in the project allowed us to prove our design efforts and increased the confidence of our client.

Usability Testing

Our team utilized Usability Testing to improve the experience of new users to our site. In order to get the most useful feedback we allowed the other team to explore the pages on our student dash and the navigation experience to determine what they liked and disliked. This feedback allowed us to make concrete improvements like a more visible navigation menu and better page layout.

Acceptance Testing

Acceptance testing was an important part of the iterative process for our team when developing the application. Our project uniquely required us to consider the perspectives of a number of users with a variety of skills and backgrounds. It was essential for us to provide our best interpretation and seek feedback to create a logical and user friendly application. During our weekly meetings with our advisor and client we found out quickly that the most valuable meetings were those where we could demonstrate our app and get feedback from Dr. Tyagi. Throughout this semester we were able to add features like in-app communications and CSV uploads. Both features make for a better user experience.

Results

The consistent theme throughout our testing is that we found new considerations we hadn't thought of before. Converting these considerations into features has added features and improved the usability.

Broader Context

Design Considerations

Public Health, Safety and Welfare
Our design will directly affect the well-being of the students by better matching them to a desired project. Faculty/advisors will also benefit from an improved web application to streamline the process. This will reduce the stress on both students and faculty by having a single application for the senior design class.

Global, Cultural and Societal
Improving the user experience during the submission of a project provides a wealth of benefits to ISU. Creating a streamlined client submission form allows a wider variety of clients to upload increasing the diversity of projects offered.

Environmental
The main purpose of this application is to help the CPRE department match students with the best project. Unlike before, this application will allow students the granularity to not only to list their preferences but their weights too. Better matching finds the best students for each project and improves efficiency and reduces waste.

Economic
Using one platform will also allow for better efficiency in terms of time spent manually assigning projects. This application will be cost effective as the only cost will be in the hosting of the application. Course administrators will spend less time on this process of the class saving man hours for the university.

Conclusions

Progress Review

Our team made considerable progress towards completing the application. Integrating the algorithm, backend, and frontend took considerable coordination and extensive testing. We also found a significant amount of work in converting the static pages left by the previous team into reactive pages. The latter took considerably more effort than expected.

Value

This project provides value to both the students and faculty of the CPRE department. Faculty benefit from the extra time and the reduced stress that the manual project matching had caused. The clients and the students benefit from the centralized interface which simplifies the project matching phase and improves the user experience. The students also benefit from better matches with the increased granularity from bidding and the matching algorithm.

Future Steps

Technical Development
Going forward we feel that the majority of the effort should be focused on improving the dashboards for currently supported users and adding new dashboards for additional user types. During the May24 term we have focused on creating the most essential use cases.

Student
The student is able to View Projects that have been submitted in the backend. The student dashboard provides a form that the student can fill out to indicate their preferences. The form is recorded in the backend for submission.

Instructor
The instructor dashboard has a lot of functionality so we have broken it into a few tabs.

The Matching Dashboard tab allows the instructor to run the matching algorithm. We would also like to incorporate metrics from the algorithm on that page. If future groups wish to enable parameters on the algorithm this would be the place for it. There is also a place to implement the Publish button if there is a desire to run the algorithm without sharing the results with the students.

The Student List tab lists all of the students in the database and provides a function to remove students. The edit button still needs implementation; we would suggest sending the user to the add student page with the current information prefilled. Improvements to the Student List page could include implementing sorting or filtering based on useful attributes.

The Modify Database tab has currently implemented the add student function. There are also frontend components for upload student roster and upload projects, these two are yet to be implemented in the backend.

Project List Tab lists all projects and provides the instructor the option to filter based on approval status. We also included buttons to approve/reject the projects and buttons to edit the project.

Client
The client form provides the client the opportunity to submit a project to the project database. The submitted project is available for all students to view.

Providing Value for Society
This project in its current stage allows instructors to perform the core operations of collecting student preferences, available projects, class management, and matching the students with their projects. Going forward we have organized the frontend and backend to be scalable so future teams can add to it without changing the architecture. The future steps section here outlines our progress and suggestions for how to get started.

Appendix 1 — Setup/Operation

Building And Running Everything

For all features of the sdmay24-02 project, you will need to run both the backend Spring Boot application and the frontend React application.

Frontend
For the frontend you will need to install all dependencies using NPM and then run “npm start”.

Backend
Use the readme in the Backend to properly configure the Spring Boot application for the local test environment.

Appendix 2 — Legacy Versions

A Brief Description of the Project We Inherited

The overall structure of the algorithm we currently have implemented is similar to what was designed in the previous phase. We have modified it slightly to conform to the new requirements we have discussed with Dr. Tyagi throughout the duration of the project.

The frontend we inherited was largely a mockup/prototype using mock data and static displays. We drew inspiration from the design but heavily modified the code to fit with best practices for organization and scalability.

There were some fragments of a backend implemented in PHP. We thought it wiser to switch to Spring Boot because it is much more in line with the coursework and our experiences in industry.

Appendix 3 — Other Considerations

Best of luck in Phase 4! Do your best, start early and explore all parts of the project to see what works best for you. It's so close!

Appendix 4 — Source Code

SD Combined Repo	https://git.ece.iastate.edu/sd/sdmay24-02
Frontend	https://git.ece.iastate.edu/nmnelson/sdmay24-02_frontend
Backend & Algorithm	https://git.ece.iastate.edu/nmnelson/sdmay24-02_backend