Transfer Pathways Tool

FINAL REPORT

Team Number: 20
Client: Susie DeMoss
Adviser: Dr. Ashraf Gaffar

Team Members/Roles:
Cameron Brecount - Co-Lead on User Interface
Ben Greif - Lead on Testing/Frontend Developer
Curt Lengemann - Lead on Middleware/Database Components
Riess Radtke - Co-Lead on User Interface
Scott Thurston - Co-Lead on Frontend
Luke Turczynski - Lead on API Management/Database Design Development
Cole Weber - Co-Lead on Frontend

Team Email: sdmay22-20@iastate.edu
Team Website: http://sdmay22-20.sd.ece.iastate.edu/

Revised: April 2022

Executive Summary

Development Standards & Practices Used

- SOLID Design Principles
    - These object-oriented principles guided our software to use best practices and led to cleaner, more reliable code. Although PHP is not fully object-oriented, it still allows for numerous object-oriented features such as classes and inheritance. So, this standard applies.
- IEEE 610 - Standard Glossary of Software Engineering
    - This standard provides definitions for every software engineering term imaginable. We referred to this standard to ensure efficient and effective communication within the team.
- IEEE 1016 - Software Design Description
    - Because this project involves creating a software design description, this standard applied, and ensured that the information displayed on the relevant diagrams is appropriately conveyed in an understandable fashion.
- IEEE 1233 - System Requirements
    - Because this standard details the methods relating to requirements engineering, this standard applied to our project. This standard helped us identify the business needs of the students and Admissions staff by determining what is currently in place and how what is in place can be improved upon.

Summary of Requirements

1. Constraints
    - 1.1 The project shall be in the form of an interactive web application.
    - 1.2 The project shall be mobile-friendly.
    - 1.3 The project shall use the Iowa State website template.
    - 1.4 The project shall be able to interact with a Workday backend, or a mocked version.
    - 1.5 Account creation for prospective students shall follow the format of an admissions prospect record.
    - 1.6 The project shall be completed by the end of the 2022 spring semester.
    - 1.7 The project shall be completed within 1,000 person hours.

2. Functional Requirements
    - 2.1 Functional Requirements Relating to All Users
        - 2.1.1 The project shall accept prospective student and administrator (admissions or academic advising staff) account types.
        - 2.1.2 The project shall display different views depending on the account type the user is logged in as.
        - 2.1.3 All project tables containing credits shall display the total number of credits in that table.
    - 2.2 Functional Requirements Relating to Prospective Student Users
        - 2.2.1 The project shall accept as input transfer courses and grades received from other universities.
        - 2.2.2 The project shall use the inputted transfer courses to determine transferability at Iowa State.
        - 2.2.3 The project shall output a four-year plan of the intended major of the prospective student in table format based on the existing four-year plan for that major.
        - 2.2.4 The project shall output a four-year plan of the intended major of the prospective student in flowchart format without prerequisite information based on the existing four-year plan for that major.
        - 2.2.5 While either the flowchart or table view is active, it shall display transferred courses crossed off along with the course name at the student's prior institution.
        - 2.2.6 The prospective student user shall be able to download a PDF file of the four-year plan of their intended major in table and flowchart format.

    - 2.2.7 The exported PDF files shall display the date of creation.
    - 2.2.8 The project shall have existing premade accounts for both administrators and prospective student accounts.
    - 2.2.9 The project shall allow the linking of the inputted transfer courses and majors to a prospective student account.
    - 2.2.10 The project shall allow for this linked data to be changed and deleted.
    - 2.2.11 The project shall allow for guest prospective student users to use the application without creating or signing into an account.
    - 2.2.12 The project shall display a disclaimer saying that this is an unofficial evaluation of transfer credits of a prospective student user, on guest prospective student user pages, and downloaded PDF files.
    - 2.2.13 The project shall display other resources for prospective students.
    - 2.2.14 The prospective student user should be able to select their preferred major.
  - 2.3 Functional Requirements Relating to Administrator Users
    - 2.3.1 The project shall allow administrator users to view data regarding individual prospective student users.
    - 2.3.2 The project shall allow administrators to download all individual student data in a format accessible by Excel.
    - 2.3.3 The project shall allow administrator users to view pertinent aggregate data regarding prospective student users.
    - 2.3.4 The project shall allow administrator users to reach out via email to prospective student users.

3. Nonfunctional Requirements
  - 3.1 The user interface shall be easy to navigate.
  - 3.2 The project shall respond within one second to a four-year plan query.
  - 3.3 The user shall be able to complete a session in under ten minutes.

Applicable Courses from Iowa State University Curriculum

- S E 319
- S E 329
- S E 339
- COM S 228
- COM S 309
- COM S 362
- COM S 363
- COM S 409

New Skills/Knowledge acquired that was not taught in courses

- PHP
- Laravel Framework
- Security Testing
- Interfacing with external APIs such as Workday and Okta
- Communication with a client
- Some team members had to learn:
    - Database Connections
    - REST APIs
    - Software Design Principles

Table of Contents

List of Figures

List of Tables

1. Team

1.1 Team Members

- Curt Lengemann
- Cole Weber
- Ben Greif
- Luke Turczynski
- Scott Thurston
- Cameron Brecount
- Riess Radtke

1.2 Required Skill Sets for Your Project

- UI Design - due to the fact the website should be user friendly for transfer students
    - ISU Website Template Knowledge - due to the fact that ISU already has a style and a template set up and in use. We must follow this template in our UI design.
- Website Design - due to the fact that the client wants a website
    - Mobile Friendly Design - due to the fact that the client has the constraint that the webapp should be mobile friendly
- Databases and Connections - due to the fact that data is stored in a database
- Software Project Management - due to the fact that we are managing a large, long term software project with many developers and stakeholders
- REST API - due to the fact that the frontend and middleware are communicating via a REST API
- Software Design Principles - due to the fact that the website needs to be modular to allow for future use and integration with Workday and Okta
- Communication - due to the number of correspondents we have
    - Communication with team members
    - Communication with the TA
    - Communication with client
    - Communication with advisor
- Source Control (GitLab) - We utilized a shared source control for the project to easily develop code

1.3 Skill Sets Covered by the Team

- UI Design -
    - All
- Website Design -
    - All
- Databases and Connections -
    - Ben Greif
    - Curt Lengemann
    - Scott Thurston
- Software Project Management -

    - All
- REST API –
    - Luke Turczynski
    - Scott Thurston
    - Curt Lengemann
    - Cameron Brecount
- Software Design Principles –
    - Curt Lengemann
    - Cameron Brecount
- Communication –
    - All
- Source Control –
    - All

1.4 Project Management Style Adopted by the Team

Waterfall with some Agile elements

1.5 Initial Project Management Roles

- Curt Lengemann - Database Engineer / Middleware Engineer
- Ben Greif - Test / Frontend Engineer
- Scott Thurston - Frontend Engineer
- Luke Turczynski - Report Manager / API Engineer
- Cole Weber - Meeting Facilitator / Frontend Engineer
- Cameron Brecount - UI / Frontend Engineer
- Reiss Radtke - Meeting Scribe / UI

2. Introduction

The Transfer Pathways Tool is an effort to create a tool with a modern user interface and more features based off the Iowa State TRANSIT system [4].

2.1 Acknowledgement

We would like to thank Susie DeMoss for her help on understanding the technicalities of four-year-plans and her feedback on our UI. We would also like to thank Ashraf Gaffar for his feedback on the technical side of our project and for his assistance on following proper software development practices. Lastly, we would like to thank Ben Carlson for his assistance with learning about ISU's implementation of Workday.

2.2 Problem Statement

The ISU Office of Admissions has determined that the current user interface for the TRANSIT system [4] is lacking in functionality and usability. Currently, TRANSIT does not have the ability to view four-year plans, which is crucial in determining degree progress and class schedules.

Additionally, TRANSIT's user interface is outdated, and its backend will no longer be supported in approximately two years when ISU transitions from UAchieve to Workday.

2.3 General Solution Approach

We solved the problem listed in section 2.2 by creating a new, more user-friendly version of the TRANSIT system that will be able to interface with the new Workday backend. Called the Transfer Pathways Tool, this application displays a four-year plan of the intended major of the prospective student at Iowa State in both tabular and flowchart format. While it does not display the degree audit view present in TRANSIT, its four-year plan view is arguably more helpful to prospective students, as many users report that the degree audit view is confusing. Functionality for administrators is expanded to include both individual and aggregate student data to allow for administrators to reach out to potential transfers.

2.4 Requirements & Engineering Constraints

1. Constraints
    - 1.1. The project shall be in the form of an interactive web application.
    - 1.2. The project shall be mobile-friendly.
        - Rationale: From ISU Admissions, more and more users nowadays are accessing websites via mobile devices.
    - 1.3. The project shall use the Iowa State website template.
        - Rationale: This is an official ISU application.
    - 1.4. The project shall be able to interact with a Workday backend, or a mocked version.
        - Rationale: ISU Admissions is transferring to a Workday backend in two years, and our application needs to integrate with it.
    - 1.5. Account creation for prospective students shall follow the format of an admissions prospect record.
        - Rationale: ISU Admissions wants the prospective accounts to be in their existing standard format.
    - 1.6. The project shall be completed by the end of the 2022 spring semester.
    - 1.7. The project shall be completed within 1,000 person hours.

2. Functional Requirements
    - 2.1. Functional Requirements Relating to All Users
        - 2.1.1. The project shall accept prospective student and administrator (admissions or academic advising staff) account types.
        - 2.1.2. The project shall display different views depending on the account type the user is logged in as.
            - Rationale: Prospective student users should not be able to access administrator data.
        - 2.1.3. All project tables containing credits shall display the total number of credits in that table.
            - Rationale: It is helpful to students to see totals, so they have an idea of how close they are to graduation.
    - 2.2. Functional Requirements Relating to Prospective Student Users

2.2.1. The project shall accept as input transfer courses and grades received from other universities.

2.2.2. The project shall use the inputted transfer courses to determine transferability at Iowa State.

2.2.3. The project shall output a four-year plan of the intended major of the prospective student in table format based on the existing four-year plan for that major.

2.2.4. The project shall output a four-year plan of the intended major of the prospective student in flowchart format without prerequisite information based on the existing four-year plan for that major.

Rationale: The flowchart is very helpful for more visual audiences.

2.2.5. While either the flowchart or table view is active, it shall display transferred courses crossed off along with the course name at the student's prior institution.

Rationale: The prior course name helps the prospective student understand which course successfully transferred, since they will be more familiar with that course than the ISU equivalent course.

2.2.6. The prospective student user shall be able to download a PDF file of the four-year plan of their intended major in table and flowchart format.

2.2.7. The exported PDF files shall display the date of creation.

Rationale: Our client is concerned that students could bring in out of date four-year plans and expect the same courses to transfer.

2.2.8. The project shall have existing premade accounts for both administrators and prospective student accounts.

2.2.9. The project shall allow the linking of the inputted transfer courses and majors to a prospective student account.

Rationale: So that the user does not have to re-enter information.

2.2.10. The project shall allow for this linked data to be changed and deleted.

2.2.11. The project shall allow for guest prospective student users to use the application without creating or signing into an account.

Rationale: While the ISU Admissions staff would prefer that prospective students create an account, they realize that some might be discouraged from using the tool if they are forced to create an account.

2.2.12. The project shall display a disclaimer saying that this is an unofficial evaluation of transfer credits of a prospective student user, on guest prospective student user pages, and downloaded PDF files.

2.2.13. The project shall display other resources for prospective students.

2.2.14. The prospective student user should be able to select their preferred major.

2.3. Functional Requirements Relating to Administrator Users

2.3.1. The project shall allow administrator users to view data regarding individual prospective student users.

2.3.2. The project shall allow administrators to download all individual student data in a format accessible by Excel.

2.3.3. The project shall allow administrator users to view pertinent aggregate data regarding prospective student users.

2.3.4. The project shall allow administrator users to reach out via email to prospective student users.

Rationale: This will allow ISU Admissions staff members to answer any questions the prospective students might have and otherwise assist them.

3. Nonfunctional Requirements

3.1. The user interface shall be easy to navigate.

3.2. The project shall respond within one second to a four-year plan query.

3.3. The user shall be able to complete a session in under ten minutes.

Rationale: If the application takes too long to use, prospective students will be more likely to give up and leave.

2.5 Engineering Standards

- SOLID Design Principles
  - These object-oriented principles guided our software to use best practices and led to cleaner, more reliable code. Although PHP is not fully object-oriented, it still allows for numerous object-oriented features such as classes and inheritance. So, this standard applies.
- IEEE 610 - Standard Glossary of Software Engineering
  - This standard provides definitions for every software engineering term imaginable. We referred to this standard to ensure efficient and effective communication within the team.
- IEEE 1016 - Software Design Description
  - Because this project involves creating a software design description, this standard applied, and ensured that the information displayed on the relevant diagrams is appropriately conveyed in an understandable fashion.
- IEEE 1233 - System Requirements
  - Because this standard details the methods relating to requirements engineering, this standard applied to our project. This standard helped us identify the business needs of the students and Admissions staff by determining what is currently in place and how what is in place can be improved upon.

2.6 Intended Users and Uses

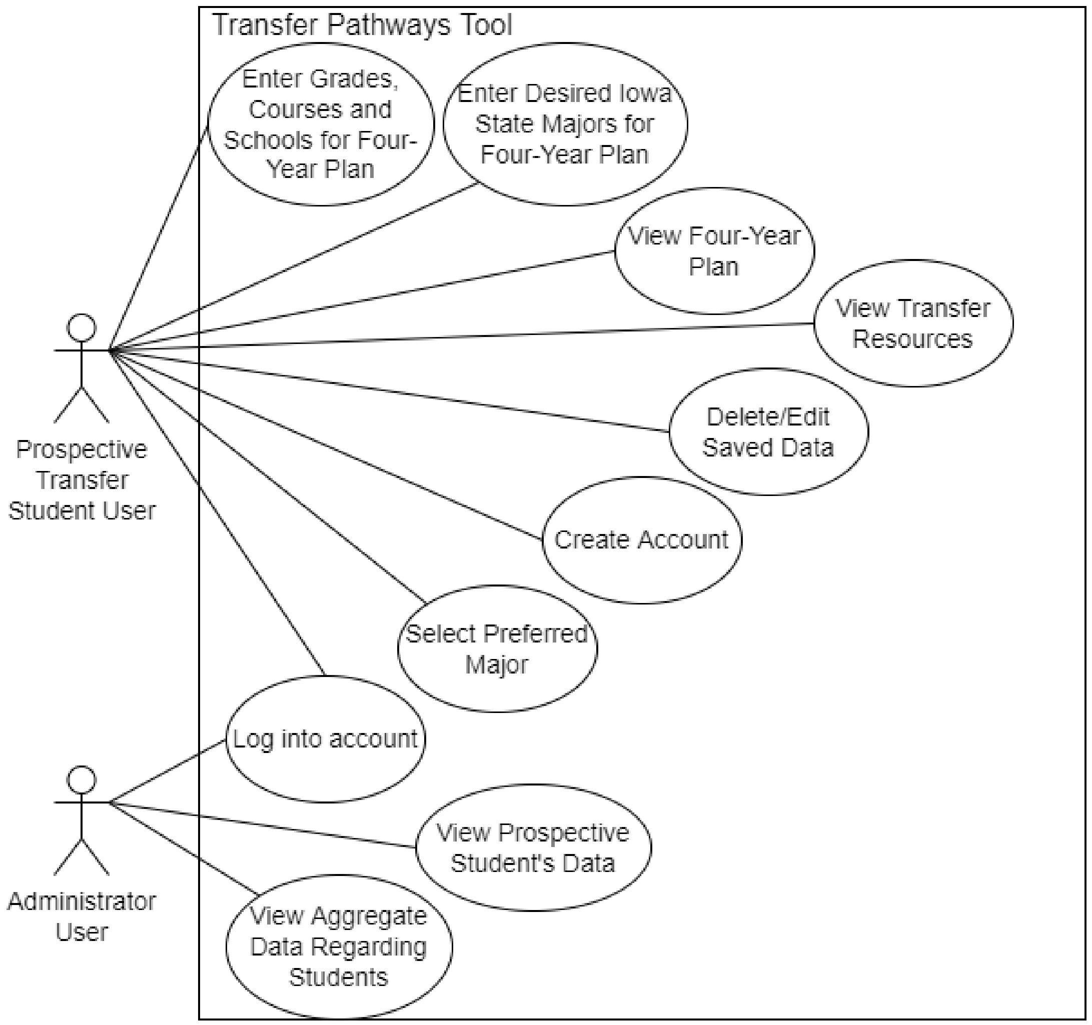

Transfer Pathways Tool
Enter Grades, Courses and Schools for Four-Year Plan
Enter Desired Iowa State Majors for Four-Year Plan
View Four-Year Plan
View Transfer Resources
Delete/Edit Saved Data
Create Account
Select Preferred Major
Log into account
View Prospective Student's Data
View Aggregate Data Regarding Students
Prospective Transfer Student User
Administrator User

Figure 1 - Use Case Diagram

- Intended User: Prospective student with transfer credits
  - The user should be able to create an account.
  - The user should be able to use the application as a guest.
  - The user should be able to sign into their account.
  - The user should be able to enter their previous courses, grades, and schools for use in computing a four-year plan.
  - The user should be able to enter their intended major at Iowa State for use in computing a four-year plan.
  - The user should be able to view a four-year plan of that major with the classes that have successfully transferred crossed off in both a flowchart and tabular format.
  - The user should be able to view previously entered intended majors and previous courses, grades, and schools.
  - The user should be able to view, edit, and delete their saved data.
  - The user should be able to view more information regarding their four-year plan.

    - The user should be able to view external transfer resources.
    - The user should be able to select their preferred major.
- Persona: Jake
    - There is a student named Jake. Jake graduated high school one year ago and knew he wanted to go to college, but was not sure what to study. So, he attended his local community college for a year to explore colleges and take general education courses. He has now decided he wants a degree in agricultural engineering at ISU. Jake wants to know which of his general education courses will transfer so he goes to ISU's Transfer Pathway Tool to evaluate his courses. Jake first chooses to create an account so he can return to his information later. Then, he is immediately directed to a page to input his courses. After filling in each of the courses he has taken, he selects agricultural engineering to see the number of credits that transfer. Out of curiosity he also selects aerospace engineering but sees that less of his credits transfer and decides to stick with agriculture and remove that major. He then selects the 'view' button to see more information on the flowchart and four-year plan of courses. Happy with the prospect of being an agricultural engineering student, Jake clicks a resource link on the side of the screen which takes him to the ISU's administration application page.
- Intended User: Administrator (ISU Admissions staff or ISU Academic Advisor) user
    - The admin should be able to sign into their account that has elevated rights.
    - The admin should be able to view specific prospective students' saved data.
    - The admin should be able to view aggregate data regarding prospective students.
- Persona: Mary
    - Transfer Pathways has an administrative user named Mary. Mary works in ISU admissions and is writing a report on interest in different majors from outside schools. She navigates to the Transfer Pathway Tool and signs in. This is the first time Mary has visited Transfer Pathways, but she does not need to create an account because she has an account with Okta and is identified as an administrator. Mary is directed to a page with aggregate data where she investigates multiple different interactive graphs to filter by time and major to see traffic regarding prospective students. Happy with the quality of data she has found, Mary uses the data she has found to create an impressive report.
- Persona: John
    - John is another administrator working for ISU, but John is an academic advisor. Like Mary, John can sign in with Okta. When he does, he is directed to the aggregate student data page. From here, he knows he is just looking to reach out to potential students. He uses the navigation to go to the student data page. John knows he wants to reach out to Jake, so he searches for that name. He finds him and clicks his email which opens his default email app so he can send Jake an email.

2.7 Assumptions and Limitations

Our project does have some assumptions and limitations largely due to the nature of the project itself. Our assumptions and limitations are as follows:

- Our product only is certain to run on Chrome and Edge browsers.
- The eventual Workday will be able to give our program the data defined by our

WorkdayConnector interface.
- IT will eventually run the middleware on a large enough server to handle all traffic as defined by our load estimate in section 5.8.
- IT will eventually maintain the project through any future updates.
- All transfer courses that do not have a direct ISU course equivalent will be matched to an ISU requirement by an academic advisor.
- IT will implement our OktaConnector interface to connect to ISU's implementation of Okta since we could not obtain access to it.
- Our security runner, SonarQube, cannot be run on CI due to the free subscription limitations.

3. Project Plan

The hybrid (both waterfall and agile) project management method, along with a detailed breakdown of tasks regarding the frontend, middleware, and backend, were used to plan this project.

3.1 Project Management/Tracking Procedures

Our group utilized the waterfall project management model with some iterative elements of Agile. We chose this model because we felt like it is more conducive to the structure of the class overall. Our first goal for the first semester was to complete a project plan and design document that were of a high quality and could be easily used as a roadmap for development of the project. We felt that we were better equipped to satisfy the goal of developing a quality project if we focused entirely on the project plan and design in the first semester, which constitutes the requirements and design phases of the waterfall method. Additionally, another goal was to develop relevant skills for the project. Our team believes that it is better to develop skills during the requirements and design phases when we have more time, than during an Agile method with sprints when we are actively trying to resolve issues. Finally, our last goal for the first semester was to ensure our project will not only function but will also wholly satisfy the needs and desires of the client. We incorporated an iterative approach to design a smooth user experience. Iterating on our screen flow, UI design, and other user-close aspects of our project ensures that we mitigate the major risk of creating a poor user experience. With constant iteration and verification with our client, we are able to meet their needs and desires for this project.

During the first semester, we utilized Discord and Google Docs to track progress. Since we mainly worked in Google Drive for the project plan document and related tasks, it made sense for us to also track progress within the same software.

During the second semester, we primarily used Agile. We had weekly meetings to create new tickets and point them so that everyone could take on an equal amount of work. This allowed us to keep track of the amount of work we could complete every week and easily track our team’s progress. We also used an iterative approach to improve our UI, also known as user centered design, the most important part of our project. We sent our UI to our client and faculty advisor to review and give us feedback on regularly so that we could constantly improve it to ensure a proper user experience. This helped us ensure that the needs of the client were met. To track progress during the development phase in the second semester, our group utilized Gitlab issues. Each issue was assigned a number of effort points, which were utilized to ensure that all group members were assigned an equal amount of work. These issues were then placed onto an issue board so that the issues were easily visible and categorized. Finally, Gitlab supports milestones as well. Each relevant issue was tied to a larger milestone. Although Gitlab was our main source of tracking progress, Discord was used as well for informal information on progress. Additionally, we used Gitlab for source control and concurrent development.

3.2 Task Decomposition

- Frontend
    - Create User Interface screen flow diagram
        - Design different components for each page
        - Articulate flow from one page to another
        - Verify UI design with Iowa State UI/UX domain experts
    - Create each individual UI component by picking high risk components first
        - Create UI for entering in transfer courses
        - Create UI for four-year plan table
        - Create UI for four-year plan flowchart - highest risk
        - Create UI for login - lowest risk
        - Create UI for prospective student account creation
        - Create UI for aggregate data screen
        - Create UI for viewing individual student data
    - Create Methods to Communicate with Middleware
- Middleware
    - Create methods to follow business logic rules to process data
    - Set up Mocked Okta API
    - Set up Mocked Workday API
    - Create REST API for middleware
- Database
    - Set up database
    - Integrate database with Mocked Workday API

3.3 Project Milestones, Metrics, and Evaluation Criteria

- Frontend
    - F1: UI Components
        - Look for components that are in Bootstrap 5 which can be used for all the functionality users will need to input their courses and output the four-year plans in an efficient UI/UX.
    - F2: Efficient Navigation
        - Be able to efficiently navigate to and from the entering courses page in less than 1 second.
    - F3: Screen Flow Interaction
        - Create screen flow diagrams that provide how each page will interact with each other.
    - F4: Middleware Integration
        - Connect frontend to the middleware using PHP and Laravel
- Middleware
    - M1: Retrieval Efficiency

        - Efficiently retrieving the four-year plans and the coursework in less than 1 second regardless of the number of courses entered.
    - M2: Mockday
        - Setting up a mocked Workday API that retrieves and stores mocked data in our database.
- Database
    - D1: Database with Workday
        - Setting up a database with mocked data in accordance with Workday, using MySQL.

3.4 Project Timeline/Schedule

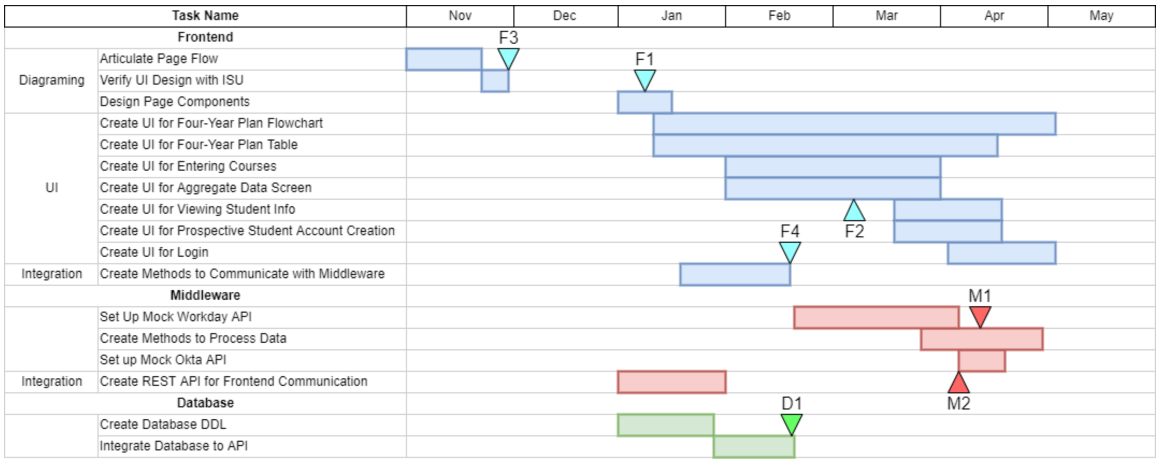


Figure 2 - Project Schedule Gantt Chart

The Gantt chart here is broken down into three major tasks: Frontend, Middleware, and Database, with the milestones indicated by the arrows. These major tasks each have their own subtasks - some of which belong to a group, like Diagramming, UI, and Integration. For this chart, we put the diagramming at the front because it is required for the rest of the Frontend work to be completed. The first semester is for documentation, while the second semester is for implementation. All tasks and subtasks are to be completed by May.

The tasks that are in parallel can be completed this way because none inherently rely on the others, and we have enough group members to accomplish each of these simultaneously.

3.5 Risks and Risk Management/Mitigation

Task	Risks & Probability (In parenthesis)	Mitigation
Create User Interface Screen Flow Diagram	None	None

Create each individual UI component by picking high risk components first	1. Components don't integrate well or make a cohesive experience (0.2)	1. Use the same framework for all components to ensure component integration
Connect Frontend to Middleware	1. Frontend and middleware cannot communicate (0.1) 2. Cannot enforce communication speed is faster than a 1 second round trip time desired threshold (0.3)	1. Use a well-known REST API framework such as Laravel to make communication between subsystems easy 2. Research other subsystem communication methods and change to a more performant one
Verify UI design with Iowa State UI/UX domain experts	1. ISU rejects the UI design (0.2)	1. Have the UI design approved incrementally to have less time reworking components
Set up Mocked Okta API	1. We have a misunderstanding of how Iowa State IT implemented the Okta API (0.1)	1. Have regular meetings with our domain expert from ISU IT to fix any misunderstandings fast
Create methods to follow business logic rules to process data	1. We have a misunderstanding of how Iowa State IT will implement the Workday API which leads to incorrectly processed data (0.1)	1. Have regular meetings with our domain expert from ISU IT to fix any misunderstandings quickly
Create REST API for middleware	1. REST API cannot handle less than a 1 second round trip time (0.3)	1. Research other process communication methods and change to a more performant one
Set up Mocked Workday API	1. We have a misunderstanding of how Iowa State IT will implement the Workday API (0.1)	1. Have regular meetings with our domain expert from ISU IT to fix any misunderstandings quickly
Database	1. We have a misunderstanding of how Iowa State IT will implement the Workday	1. Have regular meetings with our domain expert from ISU IT to fix any misunderstandings

	API causing our database data to be incorrect (0.1)	quickly
Integrate Database and Mocked Workday API	1. The database and API are incompatible or refuse to communicate (0.2)	1. Plan for the connection to the database during the process of creating the API

Table 1 – Risks and Risk Management/Mitigation

3.6 Personnel Effort Requirements

Task	**Man Hours**	**Explanation**
Create User Interface Screen Flow Diagram	50	Usability is one of the major requirements for this project, so it is important to get the design right. We need to verify the design with the team often to ensure a user-friendly experience.
Create each individual UI component by picking high risk components first	240	4-year plan (flowchart and table): 80, entering courses: 50, login/sign-up: 30, aggregate data: 50, student information: 30. Creating the UI components takes the most amount of time while hooking them up to data providers should be fairly simple.
Connect Frontend to Middleware	30	We complete basic integration early in the project and finish the full integration later on. The full integration of the middleware to the frontend requires more time and effort.
Verify UI design with Iowa State UI/UX domain experts	10	This effort estimation assumes 2 meetings and accounts for any rework from feedback we receive.
Set up Mocked Okta API	15	This class and data need to be set up in accordance with Iowa State IT's implementation of Okta.
Create methods to follow business logic rules to process data	40	This task involves taking in data from different sources such as Workday and processing it to create a student's four-year plan. This can be complicated due to the number of different majors and courses.
Create REST API for middleware	40	We create basic endpoints early in the project and finish the full integration later on. The full creation of endpoints for the middleware requires more effort.
Set up Mocked Workday API	80	This entails setting up a class to communicate with a database holding mocked Workday data. This class and data need to be set up in accordance with Workday's API.
Create Database	40	We keep records on student information, their courses, majors, and so on. We also need a database for any mocked Workday data we want to use while building and demoing the program.
Integrate Database and Mocked Workday API	20	This entails using PHP to communicate with a MySQL database to store and query data. The necessary functionality of these methods is tied to the mocked Workday API.

Table 2 – Personnel Effort Requirements

3.7 Other Resource Requirements

We utilized a virtual machine for the purposes of hosting our web application. The virtual machine hosts the middleware for our project via a WAMP server. Our virtual machine was set up by ETG and runs Windows 10. Otherwise, our group didn't require any additional resources for this project.

4. Design

The design section of this document covers all of the components of our system, the functionality of each layer, as well as some of the non-technical factors that we have considered. A number of diagrams and tables are used to describe the different aspects of our design.

4.1 Design Context

This section describes some of the broader concepts in our design, including some external factors to consider, basic user needs, and the general components that will be built.

4.1.1 Broader Context

The project we have built is for ISU and thus the university stands to be affected. The main communities affected by the project include broadly the scope of college students, but more specifically the subset of those that are interested in attending ISU. By extension, this also has an effect on the community of Ames from the increase in population and monetary spending associated with students living in Ames.

Area	Description	Examples
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., solution is implemented in their communities)	A greater allure to transfer to and thus attend ISU results in more students in Ames. Therefore, incrementally increasing the need for jobs to teach and manage those students as well as providing workers for jobs students often fill.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	The project furthers the ubiquitous social expectation for automation of processes and convenience. Additionally, the current international societal standard for automated processes is that they be regularly updated for current visual and functional intuition, which is the goal of this project.

Environmental	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.	An increased usage of an online resource provides the classic advantage of a reduction of reliance on paper documentation. Additionally, the project should reduce the need for in-person meetings with academic advisors, eliminating pollutants caused by needed transportation to attend such meetings. The associated downside is the running of the servers hosting the website and the associated power consumption needed for it. However, as this project is an improvement on an existing website, an increase in power consumption will only occur if a larger server(s) is needed.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	Being an upgrade of an existing project, the only potential economic cost is the need for more server space if the website sees a large influx of traffic. There are many potential benefits for both ISU and students, as well as rippled effects for the economy at large. When students see more benefit in attending ISU from the ability to transfer their courses, ISU will see the economic benefit from an increase in tuition. Students have also been given a better way of evaluating and utilizing the courses they have taken for usage of transfers. Higher utilization of courses taken for degrees can result in a reduction of redundant/useless spending for students. The additional access of

		funds allows students to spend more on useful spending, either in Ames (as a student) or in their local communities, resulting in economic stimulation.

Table 3 – Design Broader Context

4.1.2 User Needs

A prospective student needs a way to enter previously completed courses and receive an easy to digest four-year plan in order to successfully plan their course load at Iowa State University.

An admissions employee needs the ability to view the data and activity of prospective students in order to reach out to the student to promote Iowa State University and help them successfully choose their courses.

4.1.3 Prior Work/Solutions

The other product like ours that exists is the ISU TRANSIT system [4]. It takes classes that a student has taken at other universities as well as a chosen major for the courses to transfer for. It then shows a list of what all the input classes equate to at Iowa State for the chosen major. It also outputs all classes that do not transfer whether it is because the grade the student received for that class was too low, or if there just is no ISU equivalent of that course.

The product that our project intends to replace is the ISU TRANSIT system [4]. A user will input classes that they have taken at other universities, as well as a desired ISU major. It then shows a list of what their classes transfer to at Iowa State for the chosen major. It also outputs all classes that do not transfer, either due to the grade entered for that class being too low, or there not being an ISU equivalent course.

While our application does not exactly mirror TRANSIT, we have still certainly adopted some aspects of it, such as its general screen flow and overall functionality. We however took a different approach with the user interface and final transfer output. The biggest complaints that our application aimed to address were that TRANSIT did not look good and was not user friendly. This was explicitly requested by our client.

Very few other schools have a system like Iowa State's TRANSIT software. Texas A&M has a system called "Howdy", [5] but that system can only process one course at a time and generally lacks the complete functionality our client is looking for.

ISU IT referred us to the site that Franklin University [6] uses for their transfer students in hopes that our system would be functionally and aesthetically similar, which is its main positive benefit.

4.1.4 Technical Complexity

Our design consists of three main parts that are divided into the frontend, middleware, and miscellaneous backend services. Here are some of the components/subsystems and design challenges our team encountered ([1], [2], [3]):

1. The frontend includes the diagramming of the page flow and components that make up the user interface and user experience. In industry standards, it matches expectations of communicating how the product will look and flow before implementing it. The scientific principle of harmony, not discord, has been applied here since the development team and our client have been able to agree on how the visuals of the product looks.
2. The frontend and middleware consist of PHP, Bootstrap, JavaScript, and HTML/CSS which are all web development standards currently being used across many companies and their sites. They all work efficiently and effectively together since they're oftentimes all used together. This has been applied to the mathematical principle of using the appropriate tools strategically.
3. The middleware hosts methods that follow business logic to effectively process outside data into a consistent format for our application to use. With data coming in from various external sources, integration becomes fairly complex given that the exact specifications for such sources are unknown for this project. The engineering requirement that aligns with this is "develop and understand" since developing a workable data format and understanding how we're retrieving it in the first place is critical.
4. The middleware side of the project is where we integrate with Workday, which is a leading financial and planning system-based software. Working with Workday requires us to mock their API calls and services due to the fact that the ISU implementation of Workday will not be completed for another two years. This requirement can be classified under the engineering principle of understanding and the scientific principle of "cooperation, not individualism". The entire team was unfamiliar with the Workday API calls being used. Learning about the Workday API required understanding to work with Workday's documentation and Iowa State Admissions IT to effectively use the API calls.
5. The middleware side of the project is where we also integrate with Okta, who is currently a leading authentication account software. Much like Workday, Okta requires REST API calls to retrieve the information of user's accounts, and has to be mocked because of security concerns. This requirement can be classified under the engineering principle of understanding and the scientific principle of "cooperation, not individualism". The majority of the team was unfamiliar with the API endpoints that Okta provides. This aspect of the project required understanding and cooperation in order to effectively use Okta's services and ISU IT's instance of it.

4.2 Design Exploration

This section explores the design decisions our team has come up with while comparing and contrasting what different solutions are best.

4.2.1 Design Decisions

- We utilized a database in order to mock the Workday API for the backend of our project
- We utilized a database in order to store user activity and data.
- We used the PHP language for the middleware of our web application
- We utilized Bootstrap in our frontend, including the Select2 JQuery library for interactive dropdowns
- We utilized AJAX requests in highly interactive pages to reduce server workload and client action latency

- We implemented a login by mocking Okta SSO for the purposes of allowing transfer students to save their data for later use
- Each external service has a corresponding PHP interface so that the data source can be easily substituted for another (e.g., substituting mock Workday for the real Workday)
- Users are provided with input forms to enter grades from their transfer courses

One example of weighing our design decisions is when we utilized the lotus blossom to identify potential options for users to save data for later use. We grouped the ideas into three categories, traditional login methods, file-based methods to save data, and other methods. Within these three categories, our team came up with seven feasible options, including not implementing the feature at all, as it is only a soft goal and not a must-have requirement.

	Email preset password			Okta-SSO login	Login with Google	
	Other				Traditional	
	Do not implement				Login from scratch	
	Save as .txt			Other	Traditional	
	File-based			File-based	*A way for users to save data for later use*	
	Save as .xml					

Figure 3 - Lotus Blossom

4.2.2 Decision-Making and Trade-Off

After creating the lotus blossom, our group had an open discussion about each of the potential options. Right away, we decided against the file-based methods of saving data. Because users could easily misplace the file or forget about it, we came to the conclusion that there were better methods of saving data. Also, we decided against not implementing the feature. This was because we decided that we had the requisite time, along with the desire to meet our client's soft goal. With the remaining four possibilities, we utilized a weighted decision matrix to assist in our decision making:

Selection Criteria	Criterion Weight	Okta-SSO Login		Login with Google		Login from Scratch		Email preset password	
		Score	Total	Score	Total	Score	Total	Score	Total
Team Knowledge	0.25	2	0.5	1	0.25	3	0.75	1	0.25
Recommendation	0.4	5	2	1	0.4	2	0.8	1	0.4
Ease of Use	0.35	4	1.4	4	1.4	3	1.05	2	0.7
Total	1		**3.9**		**2.05**		**2.6**		**1.35**

Table 4 – Weighted Decision Matrix

The selection criteria included prior team knowledge, recommendation from the Admissions Department, and ease of use, as our client indicated that user-friendliness was the main goal for the project. Our team weighted the recommendation the highest, followed closely behind by ease of use. The Okta-SSO login scored high marks for both its recommendation and ease of use, however, it did not receive a perfect five for ease of use due to its two-factor authentication feature. The Google login is about as easy to use as the Okta login, however, it was not recommended by Admissions. Our next option was to build a login from scratch, while the team had prior knowledge and experience with this option, it was not recommended by Admissions. Finally, the preset emailed password was an all-around bad option for our project. From the result of the decision matrix, our project had a clear best choice, the Okta-SSO login.

4.3 Revised Design

This section goes over multiple factors in our design such as the components that were built, the general functionality of the program and some security considerations. Much of this is described with the use of diagrams to show how different components connect and the flow of data for different users.

4.3.1 Design Visual and Description

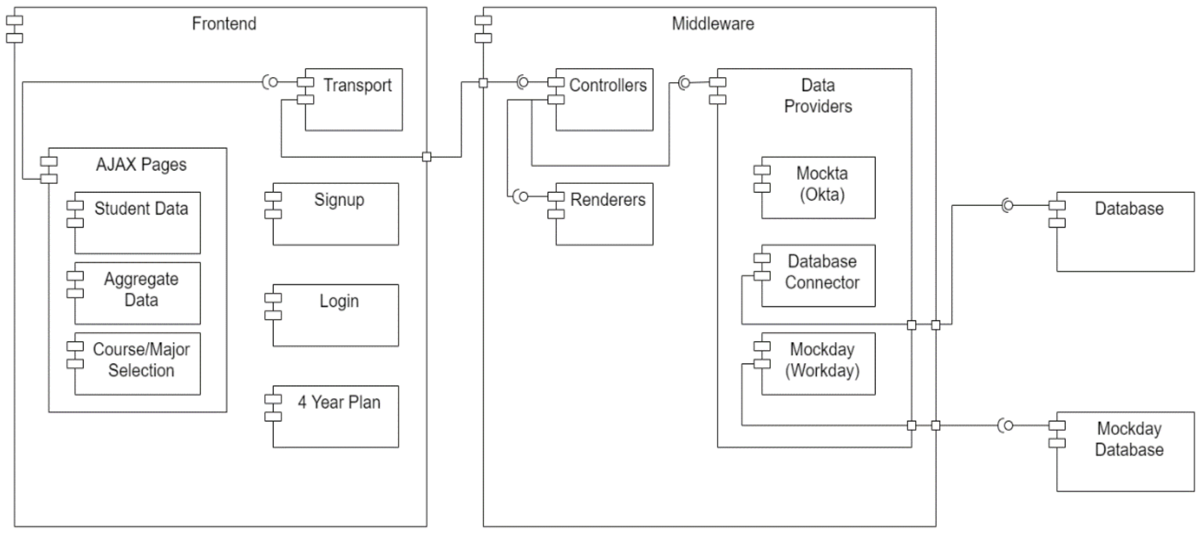


Figure 4 - Component Diagram

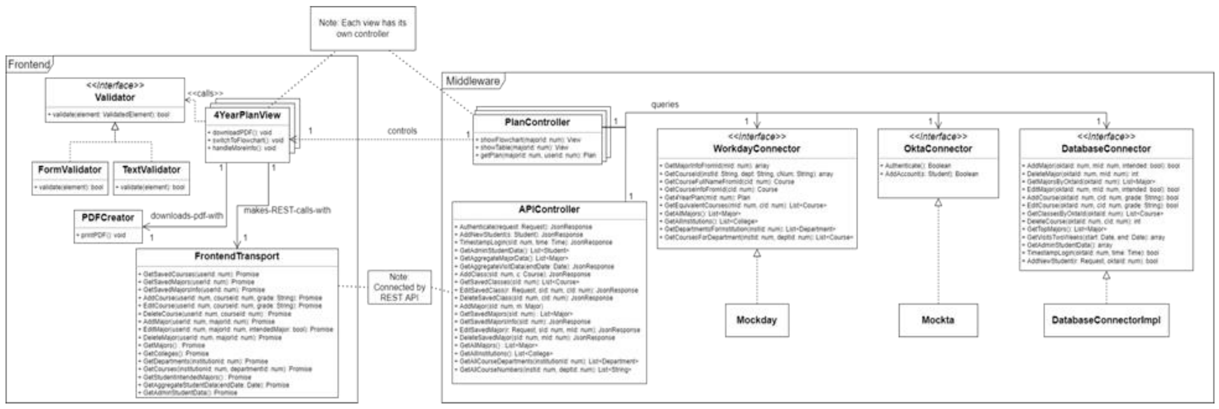


Figure 5 - Class Diagram

The two main components of our system are the frontend and middleware. We additionally have two databases: one to store user information and one to store mock Workday data. The frontend deals with user interactions and displays pertinent information. It follows a simple MVC-style design utilizing the Laravel framework. The middleware hosts REST endpoints for the various backend services.

Since the data sources are subject to change in the next few years, we have a Data Provider component to adapt the data to a consistent format that our app can use. As it stands, we are mocking the Workday API for this project: Mockday. Mockday interacts with the database for its mocked data. Additionally, the Data Provider uses a database for storing and retrieving user information. While these accounts are provided by Okta, we will need to track our app specific data for each user. This project also mocks Okta. These fake Okta users have hardcoded login credentials purely for use in testing. We have a variety of users with differing roles, where some are admin users and some are prospective student users.

4.3.2 Functionality

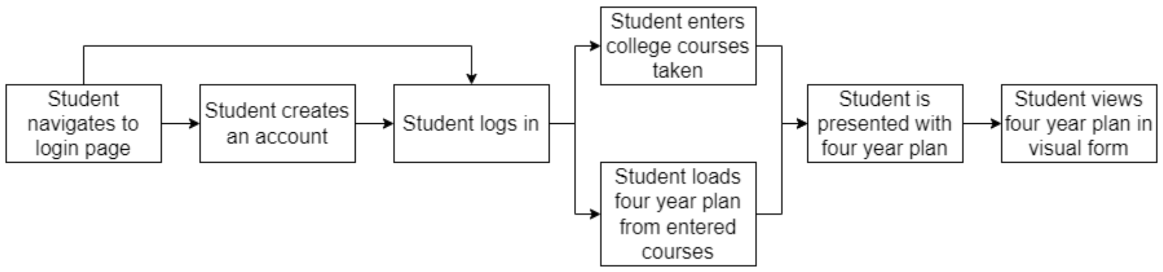

Student navigates to login page
Student creates an account
Student logs in
Student enters college courses taken
Student loads four year plan from entered courses
Student is presented with four year plan
Student views four year plan in visual form

Figure 6 - Transfer Student User Timeline

As the above Figure 6 shows, the first step for a transfer student is to navigate to the login page. Then, if they haven't created an account, they can do that next. After they have an account, they log into it. Alternatively, the transfer student may log in as a guest user. They are then presented with the choice of either entering their college courses and majors they are interested in or loading a four-year plan from previously entered courses. They then are shown their four-year plan for their selected major with the courses that they have already taken crossed out. They also have the option to view a flowchart representation of their four-year-plan.

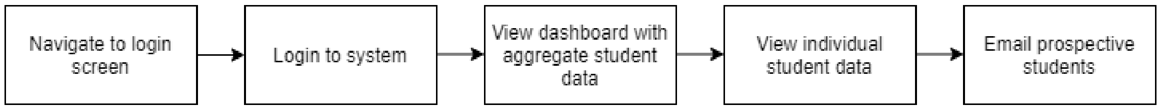

Navigate to login screen
Login to system
View dashboard with aggregate student data
View individual student data
Email prospective students

Figure 7 - Admin User Timeline

Admin users such as academic advisors and admissions employees also have to navigate to the login screen. They are able to login to the system using their Okta SSO login. Upon logging in, they are presented with a dashboard of aggregate student data such as what majors are popular for transfer students and how often the site is visited. They can then navigate to a separate page to see individual student data such as student emails. They can use that information to email prospective students to help convince them to come to Iowa State.

The current design does a nice job of satisfying the requirements for this project. All of the different functional requirements for how the system should behave have been followed. For example, administrator users can email prospective students and view student data. Another example is that transfer students can create an account, view a four-year plan of their major and view a four-year plan of their major at a later time. This design is easy to navigate because it has a very natural flow throughout the website. The design also shows that the website will be interactive so that requirement is satisfied. Lastly, this design facilitates a simple user experience so that they can spend less than 10 minutes on the site and not feel frustrated from long user sessions.

4.3.3 Design Evolution

The following section describes how our design evolved over the course of the project.

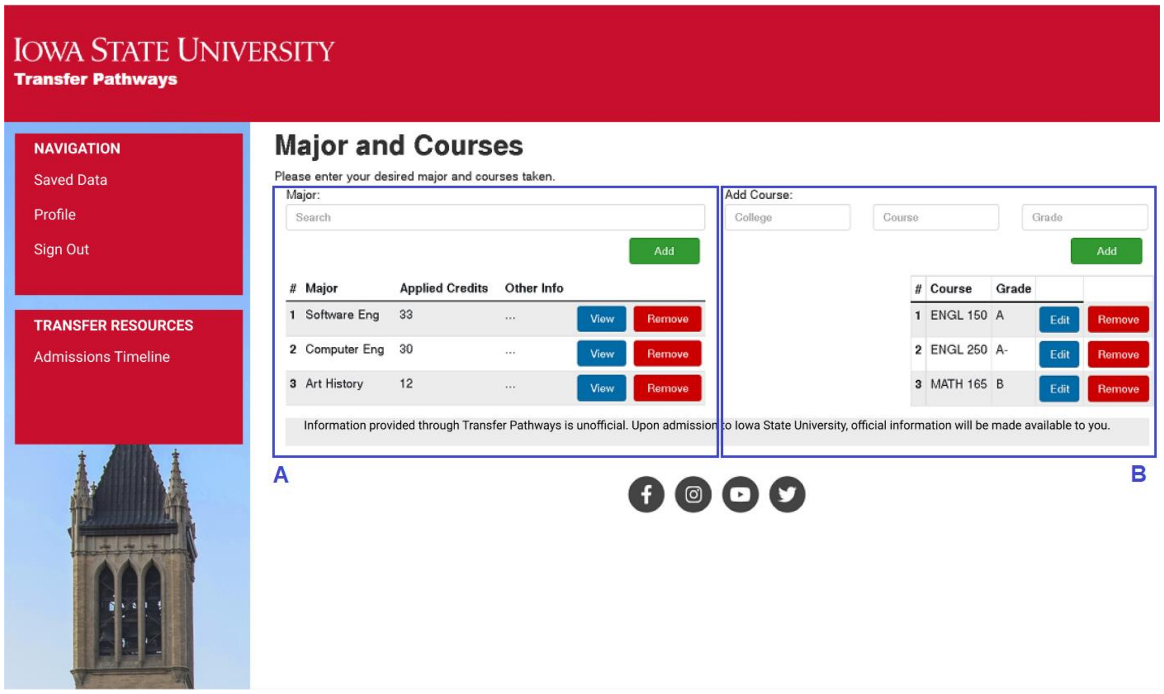
Evolution of Technical Design

Our approach for using and implementing Okta into our project was largely dependent on having access to how Iowa State University implements it on their applications. After talking with a current IT employee at Iowa State, it was discussed that it wasn't going to be guaranteed and that our group would need to contact the Okta team at Iowa State. Since we are on a short timeline and with the client's approval, we decided to omit the Okta implementation for the time being and created a mocked version of a login system. With this in mind, we have taken steps, such as using modular design principles, to make sure our mocked version can easily be swapped out for ISU's Okta implementation.

As our team better familiarized ourselves with PHP and Laravel, our design underwent some changes. We initially designed the system under the assumption that every page would use AJAX requests to update the page, and the server would not do any preprocessing. After learning more about Laravel, we quickly redesigned parts of the system to utilize its handy features, like Laravel routes, views, and controllers. The system now uses AJAX on a select few pages, while the rest are rendered purely in PHP. For more information on the specific design changes, refer to Appendix II.

Evolution Through User Centered Design

Taking a user centered approach, the UI deliberately evolved over the course of the project. We planned for and executed on several iterations to ensure that we arrived at a satisfactory UI design that meets the needs of our stakeholders. After meeting with the client to discuss our initial mockup and prototype (refer to section 6.1 for more information about the prototype), we refined the UI to better achieve the end user's goals and streamline the process of using our application. We iterated on the UI once again and repeated the process until reaching a satisfactory interface and user experience. Below shows the evolution of one of our pages: from mockup, to prototype, to initial implementation, to the final version.



Please enter your desired major and courses taken.

#	Major	Applied Credits	Other Info		
1	Software Eng	33	...	View	Remove
2	Computer Eng	30	...	View	Remove
3	Art History	12	...	View	Remove

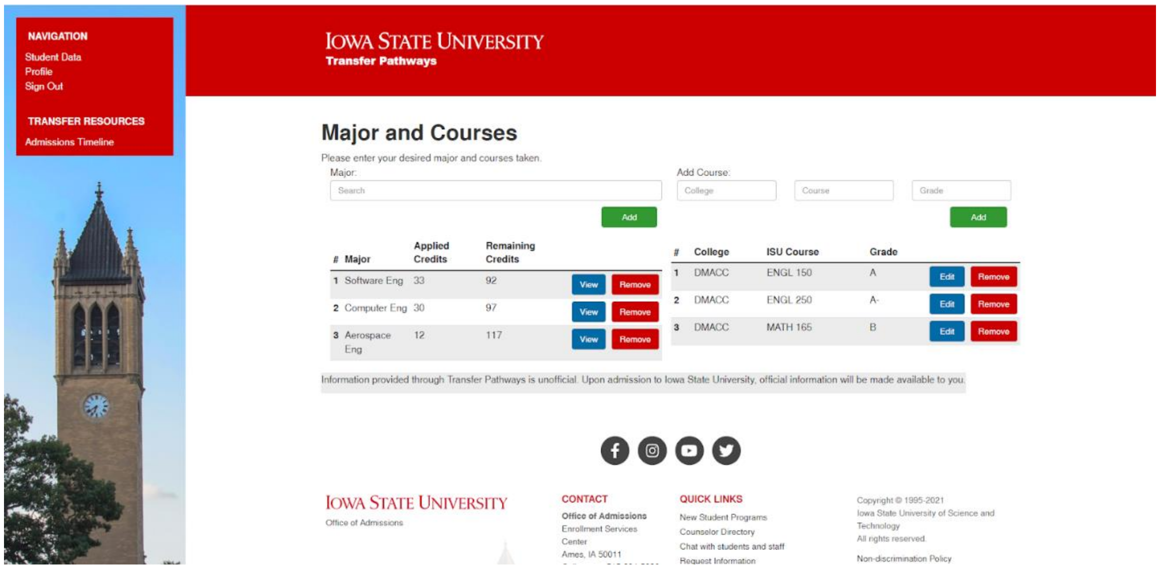
#	Course	Grade		
1	ENGL 150	A	Edit	Remove
2	ENGL 250	A-	Edit	Remove
3	MATH 165	B	Edit	Remove

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Figure 8 - Major/Courses Page Mockup

In the mockup of the Major/Courses page (Figure 8), we laid out the page in two columns: the major column (Figure 8, Label A), and the course column (Figure 8, Label B). The major column is where the student user would select an ISU major to evaluate their potential transfer to. The selected major would be added to the table. Clicking on "View" would take the user to the four-year plan generated based on the courses added in the course column.

The course column is where the student user would input their courses previously taken at other institutions. The newly inputted course would then be added to the table. Clicking "Edit" would allow the user to change the entered grade for that course.



Major and Courses

Please enter your desired major and courses taken.

Major:

Add Course:

#	Major	Applied Credits	Remaining Credits
1	Software Eng	33	92
2	Computer Eng	30	97
3	Aerospace Eng	12	117

#	College	ISU Course	Grade
1	DMACC	ENGL 150	A
2	DMACC	ENGL 250	A-
3	DMACC	MATH 165	B

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Iowa State University
Office of Admissions

Office of Admissions
Enrollment Services
Center
Ames, IA 50011

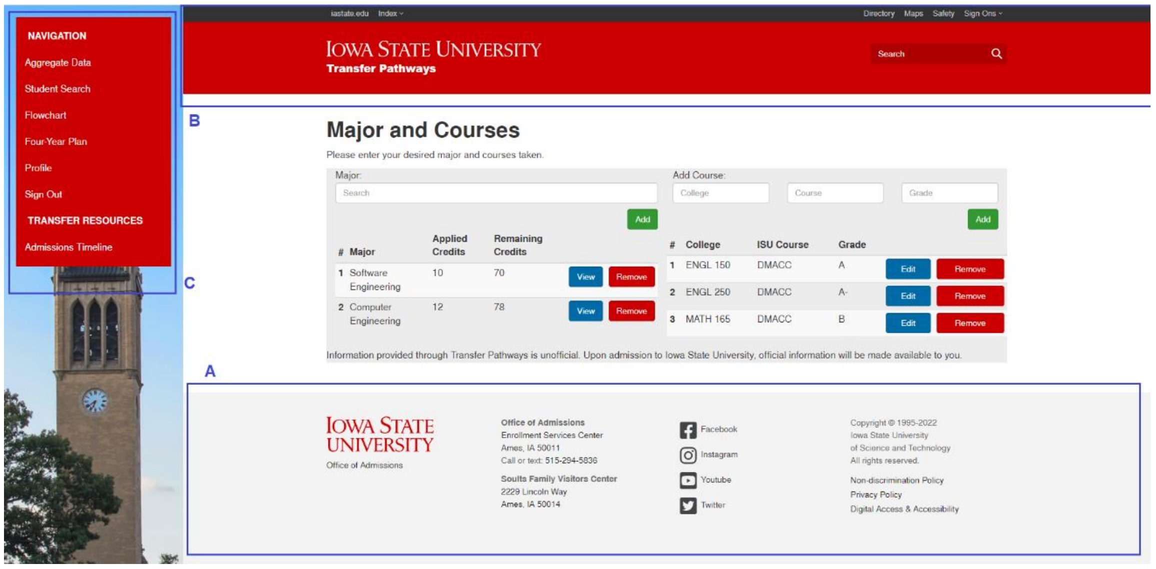
New Student Programs
Counselor Directory
Chat with students and staff
Request Information

Copyright © 1995-2021
Iowa State University of Science and Technology
All rights reserved.

Non-discrimination Policy

Figure 9 - Major/Courses Page Prototype

From the mockup to the prototype, we retained the basic layout with the only significant change being the addition of the ISU links at the bottom. These were added by the client's request.



Major and Courses

Please enter your desired major and courses taken.

Major:

Add Course:

#	Major	Applied Credits	Remaining Credits
1	Software Engineering	10	70
2	Computer Engineering	12	78

#	College	ISU Course	Grade
1	ENGL 150	DMACC	A
2	ENGL 250	DMACC	A-
3	MATH 165	DMACC	B

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Iowa State University
Office of Admissions

Office of Admissions
Enrollment Services Center
Ames, IA 50011
Call or text: 515-294-5836

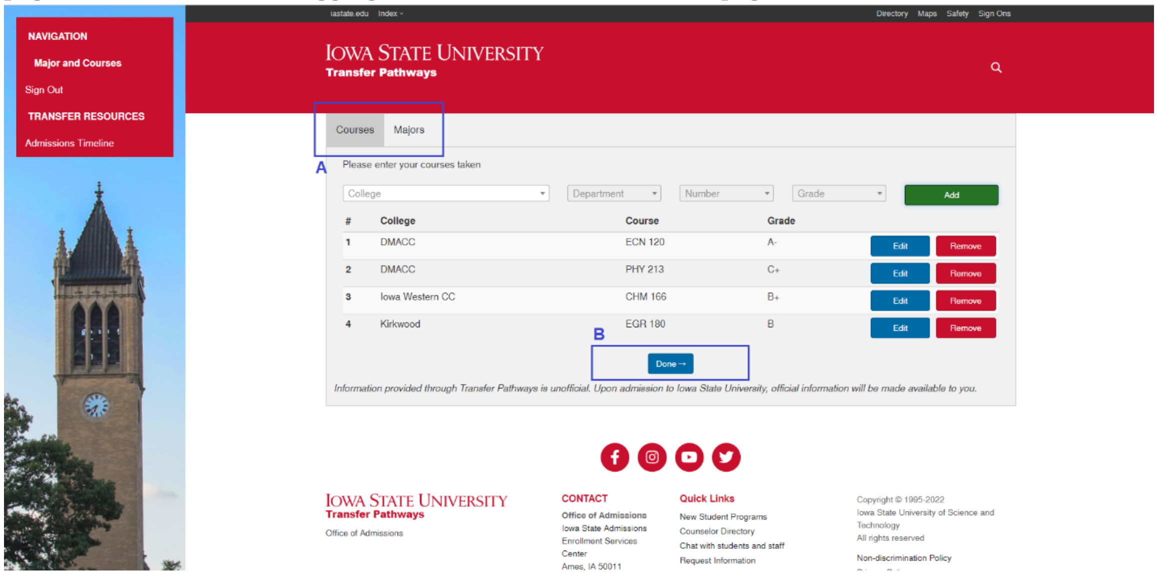
Soults Family Visitors Center
2229 Lincoln Way
Ames, IA 50014

Copyright © 1995-2022
Iowa State University
of Science and Technology
All rights reserved.

Non-discrimination Policy
Privacy Policy
Digital Access & Accessibility

Figure 10 - Major/Courses Page Initial Implementation

The initial implementation of the Major/Courses page (Figure 10) was built on top of an older version of the ISU theme, so the footer (Figure 10, Label A) was slightly different from the previous iteration. Consequently, the header (Figure 10, Label B) had slight differences. The side navigation (Figure 10, Label C) also received some changes this iteration. More links were added to pages like the Flowchart, Aggregate Data, and Student Search pages.



Please enter your courses taken

#	College	Course	Grade
1	DMACC	ECN 120	A-
2	DMACC	PHY 213	C+
3	Iowa Western CC	CHM 166	B+
4	Kirkwood	EGR 180	B

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

IOWA STATE UNIVERSITY Transfer Pathways
Office of Admissions

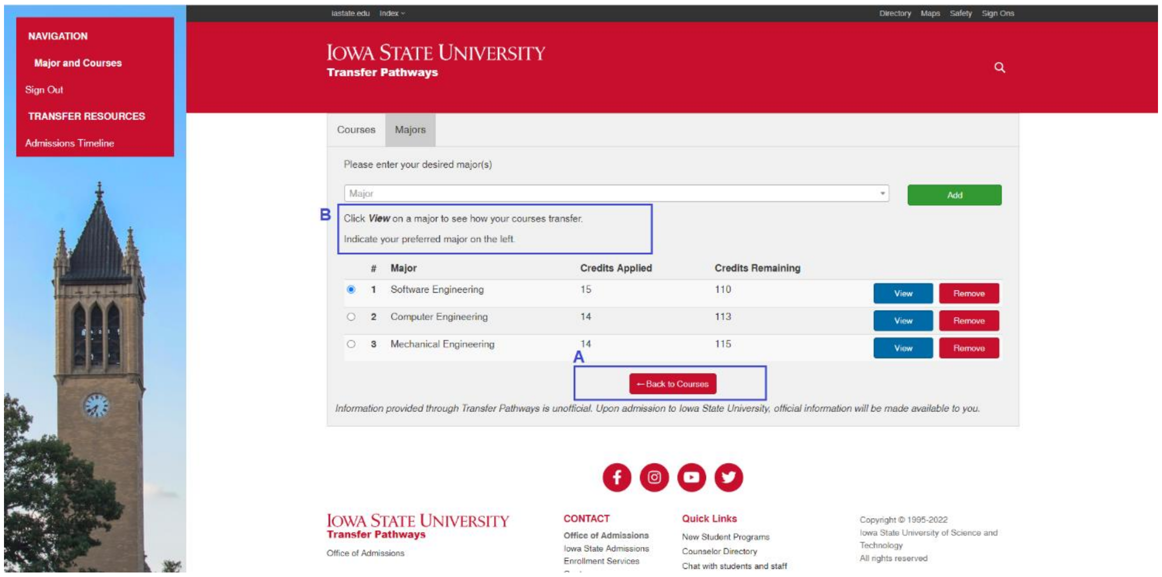
CONTACT
Office of Admissions
Iowa State Admissions
Enrollment Services
Center
Ames, IA 50011

Quick Links
New Student Programs
Counselor Directory
Chat with students and staff
Request Information

Copyright © 1995-2022
Iowa State University of Science and Technology
All rights reserved

Non-discrimination Policy

Figure 11 - Major/Courses Page Final Version - Course Tab



Please enter your desired major(s)

Click *View* on a major to see how your courses transfer.
Indicate your preferred major on the left.

	#	Major	Credits Applied	Credits Remaining
●	1	Software Engineering	15	110
○	2	Computer Engineering	14	113
○	3	Mechanical Engineering	14	115

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

IOWA STATE UNIVERSITY Transfer Pathways
Office of Admissions

CONTACT
Office of Admissions
Iowa State Admissions
Enrollment Services

Quick Links
New Student Programs
Counselor Directory
Chat with students and staff

Copyright © 1995-2022
Iowa State University of Science and Technology
All rights reserved

Figure 12 - Major/Courses Page Final Version - Major Tab

The final version of the page (Figure 11 and Figure 12) uses an updated ISU theme version, so the footer looks more like the original. One major change from the previous iteration is that the majors and courses columns were split into two separate tabs (Figure 11, Label A). The previous iterations showed that the column layout cluttered the screen and could be confusing for users. To make the process of adding courses and viewing four-year plans clearer for users, a button was added on each tab that navigates to the other tab for additional discoverability (Figure 11, Label B) (Figure 12, Label A). More instructional text was also added to the majors tab (Figure 12, Label B) so the user may better understand how to use the site.

4.4 Technology Considerations and Uses

- Workday API
    - Strengths - This is the software the University will be using as its backend API for its websites.
    - Weaknesses - Since the ISU implementation of this API hasn't been created yet, we will have no way to verify what the output of the API will be like, forcing us to rely on ISU IT to implement methods we need.
- Okta API
    - Strengths - Okta provides a quick and easy way for people with Okta accounts to sign in. Additionally, the university already uses Okta.
    - Weaknesses - New Okta accounts will need to be created for prospective students since they will not already have one. For security reasons, our group did not have access to ISU's Okta system and had to create a mocked version for testing purposes.
- PHP
    - Strengths - This is the language used by ISU for its websites. Using it allows our project to better integrate with the websites that already exist and allows us to use the University's web development templates.
    - Weaknesses - Not all of our team members had experience with this language, which added a learning curve to our project completion time.
- PHPUnit
    - Strengths - PHPUnit is the premiere testing software for PHP code, it is incredibly powerful and can get everything done.
    - Weaknesses - Only a few of our team members had prior experience with PHPUnit.
- JavaScript
    - Strengths - JavaScript enables our pages to be more interactive and responsive. Offloading some of the processing tasks to the client reduces the server's workload and decreases the latency between a user's action and the user seeing an update on the page.
    - Weaknesses - Adding another language to our tech stack increases the complexity of the project and the likelihood of system faults.
- Laravel Dusk
    - Strengths - Laravel Dusk is a powerful framework for testing web applications.
    - Weaknesses - None of our team members have used it in the past. It also adds considerable processing time to our CI system.
- SonarQube
    - Strengths - SonarQube is a powerful tool for automated security testing.
    - Weaknesses - No one in our group has used SonarQube in the past.
- Virtual Machine
    - Strengths - A VM can easily host our frontend/middleware communication API.
    - Weaknesses - We needed to find a way to host the VM on a university server.

4.5 Design Analysis

The proposed design as in Figure 5 - Class Diagram been verified and understood by the client and IT professionals from ISU. The visual diagrams have been properly thought out to encompass the larger focus of this project which is the frontend. The middleware has been created in order to properly interact with the 3rd party software that ISU IT and our client want us to use. Using this design, we can eventually utilize ISU's implementation of Okta to authenticate users and integrate with the eventual Workday backend. This design was also created with the intent of ensuring our system is extremely modular and could be integrated with any number of databases and active directory systems. This also ensured that integration with Workday would be easy for ISU IT down the road. The class diagram provides specific methods and classes that were used for the project that depict the modular design of our system. The timelines provide insight to the flow of how users will use the system and which mimics what's already in place along with the additional requirements to this project.

Iterating over the transfer student user experience was important for our project since it is the most important aspect of our project. Since there are many steps to the transfer student timeline, we needed to make sure each one was well done and verified by the client to fulfill the requirements of the project. Like the student timeline, the admin timeline needed to be iterated over since providing ISU Department of Admissions employees with adequate data is crucial to potential prospective students. We feel the current design is satisfactory due to its modular nature and won't need to be altered much for future development.

4.6 Development Process

As stated previously, our team followed the waterfall management model with some elements of Agile. As we transitioned from the design phase to the development phase, we took on a more Agile-like style. We followed a user-centered design methodology by developing a barebones UI wireframe to be verified by the client. The client critiqued the initial wireframe and our team iterated on some aspects of the design. After the client signed off on the new functionality and user experience, we transitioned to a "beautify" phase, where we transformed the wireframe into a good-looking interface. This "beautify" phase involved making the UI look like the previously approved UI mockup and making it mobile friendly. Again, the application was sent to the client for review, we iterated on the design, and finally, the client approved.

Each development task was partitioned into tickets and pointed based on our effort estimations. The tickets were categorized into frontend, middleware, database, documentation, testing, bug, and security. Tickets were prioritized based on our plan timeline and assigned based on member interest, role, and point distribution. After completing a ticket, the team member would create a merge request. We required that at least one other member review and approve the changes. For particularly risky or technically difficult tickets, we required more than one reviewer.

4.7 Security Concerns and Countermeasures

Security is becoming more and more important in software development than ever before. We made sure to take this into consideration when developing our web application and we completed some countermeasures to keep our software physically secure and to promote cybersecurity.

4.7.1 Physical Security

While our project is a web application so many physical security countermeasures are not required, we did back up all of our database data to Google Drive in order to prevent data loss in the event that our server crashes or something happens to it. This allows for every user to be able to maintain all data by just copying the backup data to a new database in the case of a natural disaster or server crash.

4.7.2 Cybersecurity

Cybersecurity was a large emphasis in our project and we made sure to consider it in every aspect of the product's lifecycle. We were deeply concerned about malicious users getting access to data they shouldn't have or causing a denial-of-service attack on our website so we took some countermeasures to prevent these from happening.

First, we installed the security testing tool SonarQube to scan through our code and inform us of any security vulnerabilities that existed in our project. We were able to fix all of the vulnerabilities that it found. We first fixed resource integrity issues with some JavaScript import statements to ensure checksums were present to validate the authenticity of all scripts. Next, we fixed an issue where our regex wasn't being evaluated in linear time. We put a character limit on the input to prevent denial of service attacks. Finally, we made our project only allow cross-origin resource sharing of our API calls with our frontend. This keeps any third-party users from utilizing our API and getting access to any data.

Next, we enforced that all queries that we made were parameterized queries. This ensures that our project is safe from SQL injection attacks. This is very important so a malicious user cannot get access to our database and steal, edit or delete data.

Then, we created input validation on all user text input. This allowed us to perform input sanitization to keep users from inputting malicious inputs that will crash the server or perform other attacks. This was very important for resource integrity and reliability.

Finally, we encrypted all stored session data on the server. This ensures that even if a hacker gets access to the server, they will not be able to retrieve the user session data. This helps protect all user data.

5. Testing

To ensure our program was of high quality and running smoothly, we developed a plan for testing our software from all different angles. Testing was performed throughout the entire course of development using different methods and tools to ensure that our web application worked as intended.

5.1 Unit Testing

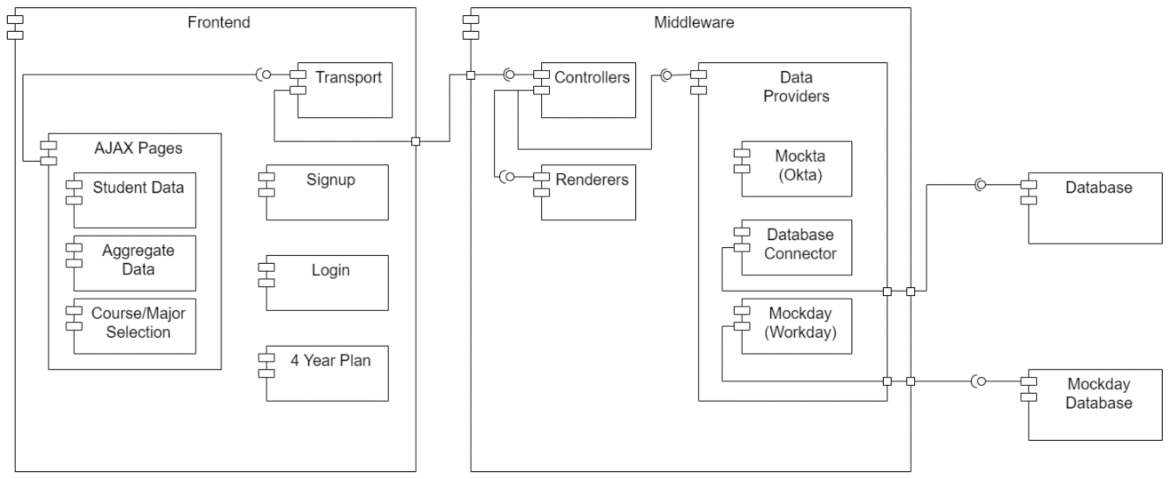


Figure 13 - Component Diagram

The above diagram shows the different components of our design. Many of these components have unit tests.

Unit tests for the server-side of our web application were conducted using PHPUnit. PHPUnit was selected as our unit test framework because it is the most popular framework for PHP unit tests, it is actively maintained, and it works well with our Laravel project. All of our adapted models (models that come from some arbitrary source but are adapted to one interface that our app can use) had unit tests, along with other testing methods as well. These included the CourseTest class, which tested the adapted Course class, which itself had numerous functionalities such as making course maps and getting applicable courses. Additionally, the test class MajorCoursesTest contained a test of the Majors adapted class. In addition to the extensive unit testing of our adapted models, we had a unit test to verify the correctness of our Laravel theme information, such as the site title and external links.

On the client side, we had many unit tests for our validators using Mocha. We chose Mocha as our JavaScript unit testing framework because some of the group members had prior experience with it. These validators were used for validating user inputs in our Sign-Up page, such as email inputs and required field inputs. All validators in our application had unit tests which made out to 38 tests in total. This is one instance of our testing strategy ensuring confidence in our system. When applied in combination with our other testing methods to the rest of the system, we can be confident in the correctness of the system as a whole.

5.2 Interface Testing

In terms of interface testing, our API received the bulk of tests. Every single one of our API routes had an API interface test located in the class APITest. These API tests were what Laravel calls “feature tests,” which utilize PHPUnit as a test runner but are able to interact with a larger part of the Laravel framework than unit tests. The general format of each API test is sending an HTTP request to the specified API endpoint to be tested, check that the status came back as 200 (OK), and check that the request data is as expected. For example, in the testGetAdminStudentData method, we insert three rows of data into our middlewaredb database, one containing information for a test user with email TEST@gmail.com. Then, we send an HTTP request to our API route for getting admin student data with no parameters. Next, we check that the response had a status code of 200, and that it contains “TEST@gmail.com”, among other things. Finally, we delete the temporary data in the database for this test. Another example of interface testing in this project was our WebRoutesTest class, however, due to the fact that we protected many of the pages based on user type, only two of our routes were included in this test. More extensive testing of these protected pages was done using unit and system testing.

On the manual testing side, we used Postman to make API calls to the running middleware. Postman was used as a stand-in for the frontend so we could isolate both the call and the response. This was helpful in both writing API documentation and for troubleshooting when the result of an API request was not as expected.

5.3 Integration Testing

To test the integration between different components in our web application, we wrote some integration tests to cover the interaction between our databases and their models, along with four-year plan calculation. These tests were written in PHP and run using PHPUnit. The class DatabaseIntegrationTest contained many testing methods in order to verify the integration of our database models and the two databases in our application, workdaydb and middlewaredb. All models and database tables in both of our databases were covered within this test class. Another instance of integration testing in our application was the FourYearPlanTest class. The purpose of this integration test was to verify the correctness of the four-year plan calculation functionality. This functionality is handled by the PlanController class, which itself requires interaction with many different classes (and in turn both workdaydb and middlewaredb). The FourYearPlanTest is arguably one of our most important tests, as the four-year plan calculation functionality is one of our most important requirements.

5.4 System Testing

For system testing, our group used automated functional tests written in Laravel Dusk, which uses PHP-Webdriver under the hood. We chose Laravel Dusk because it easily integrated with our existing Laravel project. Additionally, the Chrome Developer Tools were used to assist with web scraping our application’s elements. In total, 24 system tests were created, and these tests covered all webpages of our site. Of those 24 tests, we picked three critical features that received extensive functional test coverage. These critical features were the ability for the user (a prospective transfer student) to input transfer courses and grades, the ability for the user to view a four-year plan for the Iowa State courses in their intended major (with the courses they have transferred crossed off), and the ability to display different views depending on the account type the user is logged in as. By using extensive system testing for these critical features, we were able to ensure our main requirements were satisfied, and that they were not regressed by future code changes. These automated tests were system level tests because they interact with the system as a whole, in our case a web application, from the perspective of an admin or prospective student user. In addition to automated tests, manual system testing was utilized by both the developer and sometimes reviewers as well depending on the merge request.

5.5 Regression Testing

To ensure that new features or bug fixes did not break our existing functionality, our group primarily relied on GitLab's CI/CD feature. Early on in our project we configured a CI/CD pipeline for our repository. This pipeline ran all unit tests, integration tests, and system (functional) tests using a Docker image build specifically for Laravel Dusk. Our pipeline ran both when a merge request was created (the merge request was prevented from being accepted until all tests passed), a merge request was updated with a code change, and after a merge request was merged into the main branch. By running all our tests both before and after merge, our ability to identify and fix regressions depended on our test coverage. Since we had sufficient unit and integration test coverage along with large system (functional) test coverage for our main requirements as mentioned in the prior section, this was sufficient to identify nearly all regressions. However, when we identified a regression that was missed by the CI/CD pipeline, we created a ticket to fix the issue and created a new test to identify the regression in the future. Finally, manual testing was utilized as well by both the author of the merge request and reviewers if there was a high regression probability identified or if the author of the merge request requested manual testing.

The critical features that we needed to ensure did not break were the ability for the user (a prospective transfer student) to input transfer courses and grades, the ability for the user to view a four-year plan for the Iowa State courses in their intended major (with the courses they have transferred crossed off), and the ability to display different views depending on the account type the user is logged in as. These critical features were derived from our project's functional requirements. Since these use cases were heavily tested, we did not have any major regressions in these areas during development.

5.6 Acceptance Testing

For each issue created in Gitlab (68 in total were merged over the course of the project), the author attempted to identify very specific acceptance criteria. It was the job of the developer to satisfy this acceptance criteria and the job of the reviewer of the merge request to ensure that the acceptance criteria were satisfied. The acceptance criteria mainly dealt with functional requirements. In terms of non-functional requirements, the performance of page and item load times were evaluated by manual testing to verify acceptance. Other non-functional requirements were tested according to their specific acceptance criteria.

Although the process of validating our web application was successful, we needed to verify the end-result with our client. Every few weeks we held demos with our client for final acceptance testing. These demos were highly beneficial since our client was able to see our progress and was also able to use the application herself if she wished. Since we held multiple demos before the final demo, we were able to fix any issues our client identified more quickly than if we had waited to demo our final product. By the second to last demo, only minor issues were identified by the client. At the final demo, we were able to showcase a polished final product without the need for quick fixes.

5.7 Security Testing

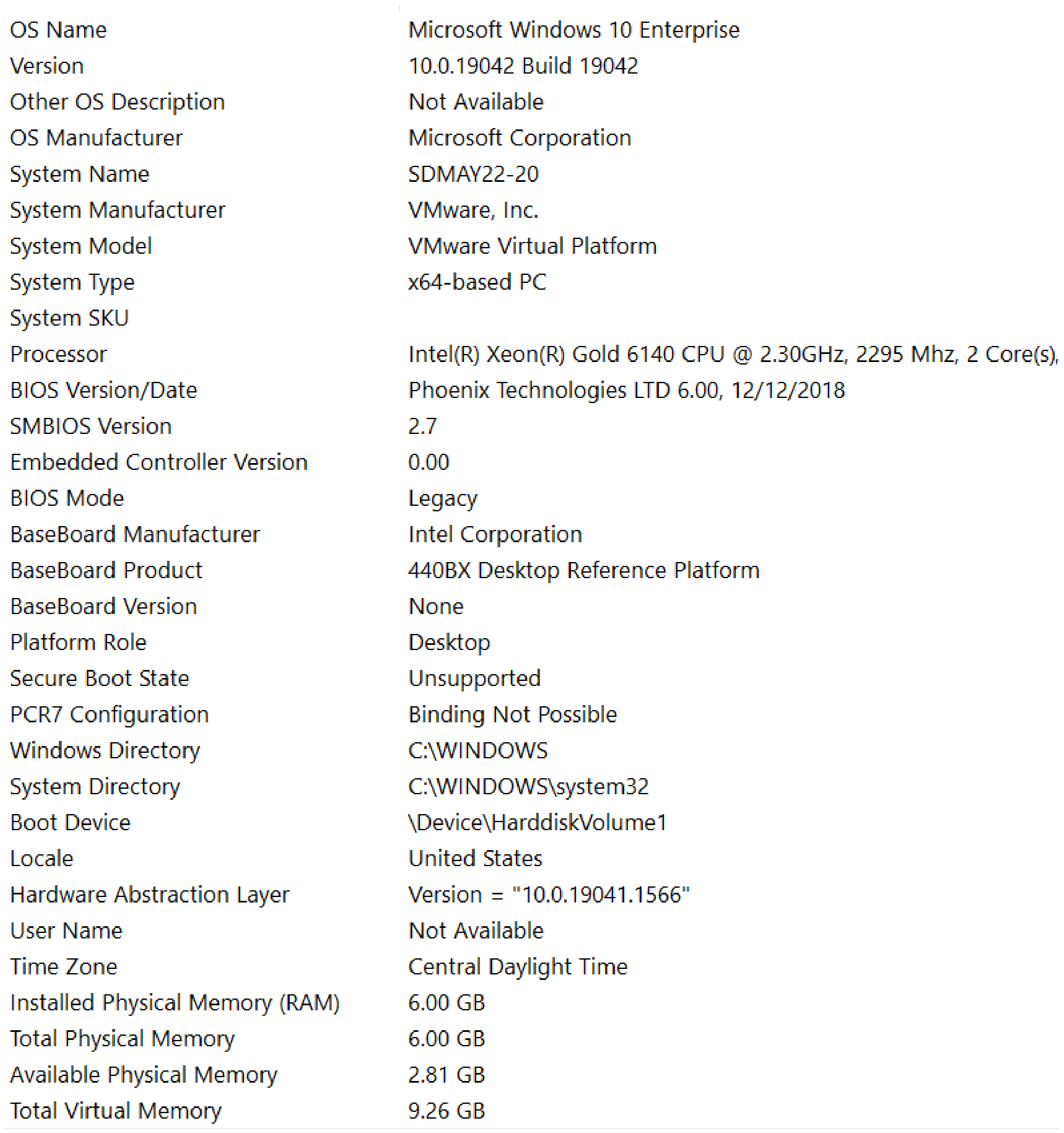
Our group used SonarQube and the SonarQube Scanner to run security analysis on our web application. SonarQube was chosen because it has a Community Edition that is free and because it runs code analysis and is not limited to dependency analysis. We utilized the SonarQube

Community Edition and provided instructions on how to set up and run both the software and the scanner. While SonarQube has many different features, our group focused exclusively on what SonarQube calls "Security Hotspots." Security Hotspots range in priority and can even be ignored if developers think they are not an issue. In our initial analysis of our web application, four categories of vulnerabilities were identified. First, we had http and not https links to certain external sites such as the Iowa State Admissions Facebook page. This was an easy issue to fix and was fixed immediately. Second, one of our regex expressions was identified by SonarQube as potentially running in non-linear time. The regex was fixed by adding in character limits; this resolved the issue. Third, we utilized some external JavaScript libraries such as Chart.js without adding resource integrity hashes. We used srihash.org to generate integrity hashes which resolved these issues. Finally, SonarQube flagged our Cross-Origin Resource Sharing policy. By updating our allowed origins to only include our site, this issue was resolved. In total, most Security Hotspots identified by SonarQube were resolved, except for one link in our site which could not be changed to https. While SonarQube proved successful in making our site more secure, one limitation was the fact that the Community Edition did not allow us to run SonarQube in our CI/CD pipeline. Running SonarQube in our pipeline would have allowed us to better prevent security regressions.

5.8 Load Testing

Load testing for our website was conducted using ApacheBench. This decision was made because of the good compatibility it has with our Apache Web Server and because many load testing programs are very expensive while ApacheBench is free and open source. We chose to conduct load tests rather than stress tests because we wanted to access the performance of the system and find the upper limit of users for our application. Since stress tests use an extreme load and test the robustness of the application rather than the performance, we felt that load tests would provide us with more useful information in this case.

Here is the system information for our web server:



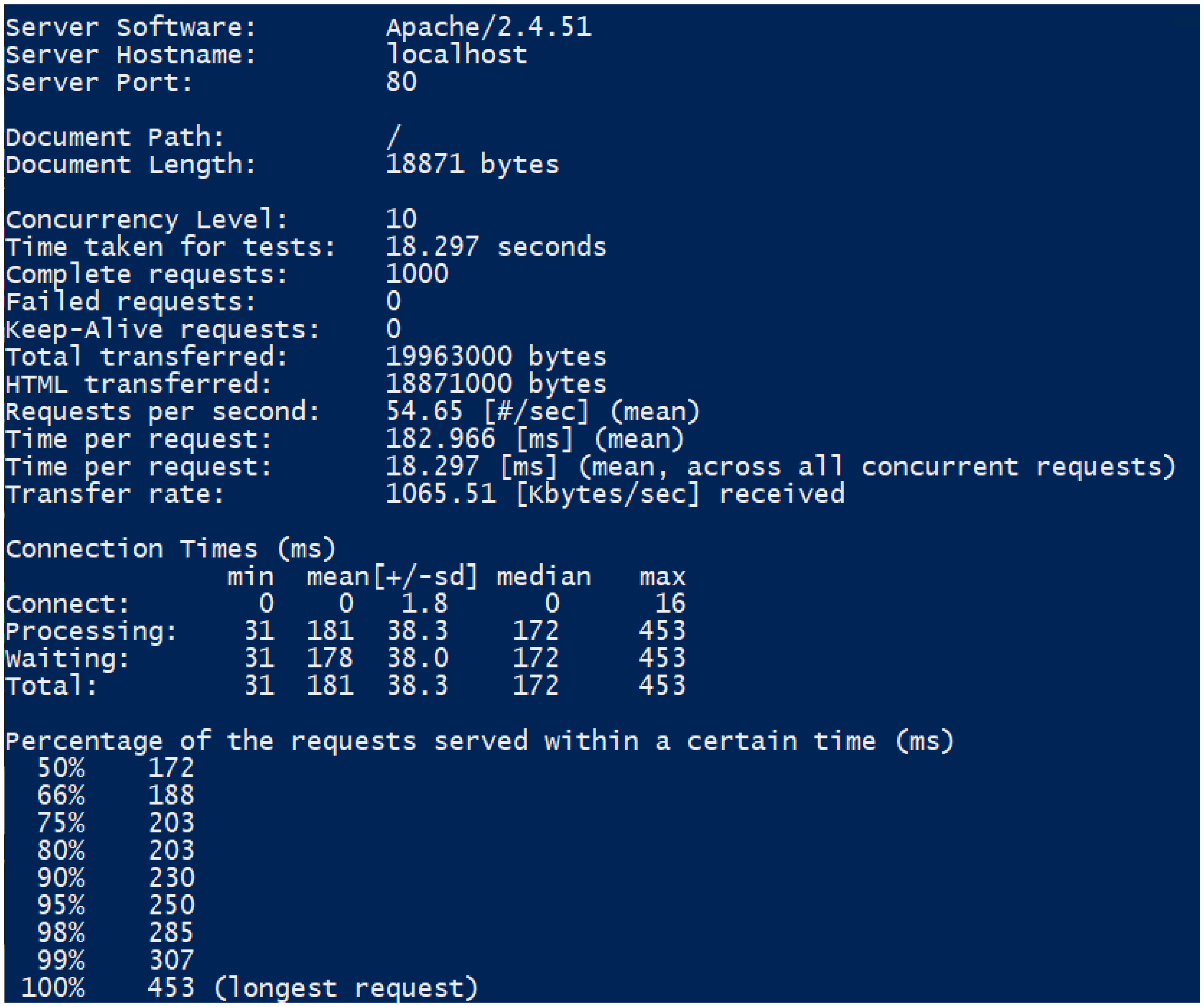
OS Name	Microsoft Windows 10 Enterprise
Version	10.0.19042 Build 19042
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	SDMAY22-20
System Manufacturer	VMware, Inc.
System Model	VMware Virtual Platform
System Type	x64-based PC
System SKU	
Processor	Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz, 2295 Mhz, 2 Core(s),
BIOS Version/Date	Phoenix Technologies LTD 6.00, 12/12/2018
SMBIOS Version	2.7
Embedded Controller Version	0.00
BIOS Mode	Legacy
BaseBoard Manufacturer	Intel Corporation
BaseBoard Product	440BX Desktop Reference Platform
BaseBoard Version	None
Platform Role	Desktop
Secure Boot State	Unsupported
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.19041.1566"
User Name	Not Available
Time Zone	Central Daylight Time
Installed Physical Memory (RAM)	6.00 GB
Total Physical Memory	6.00 GB
Available Physical Memory	2.81 GB
Total Virtual Memory	9.26 GB

Figure 14 - Server System Information

Additionally, we are running a WAMP Server with Apache version 2.4.51, PHP version 7.4.26, and MySQL version 8.0.26, and all tests were run using the HTTP keep-alive flag. Additionally, our API is rate-limited at 120 requests per minute per user.

5.8.1 Home Page

First, the home page was tested with 1000 total requests and 10 concurrent requests.

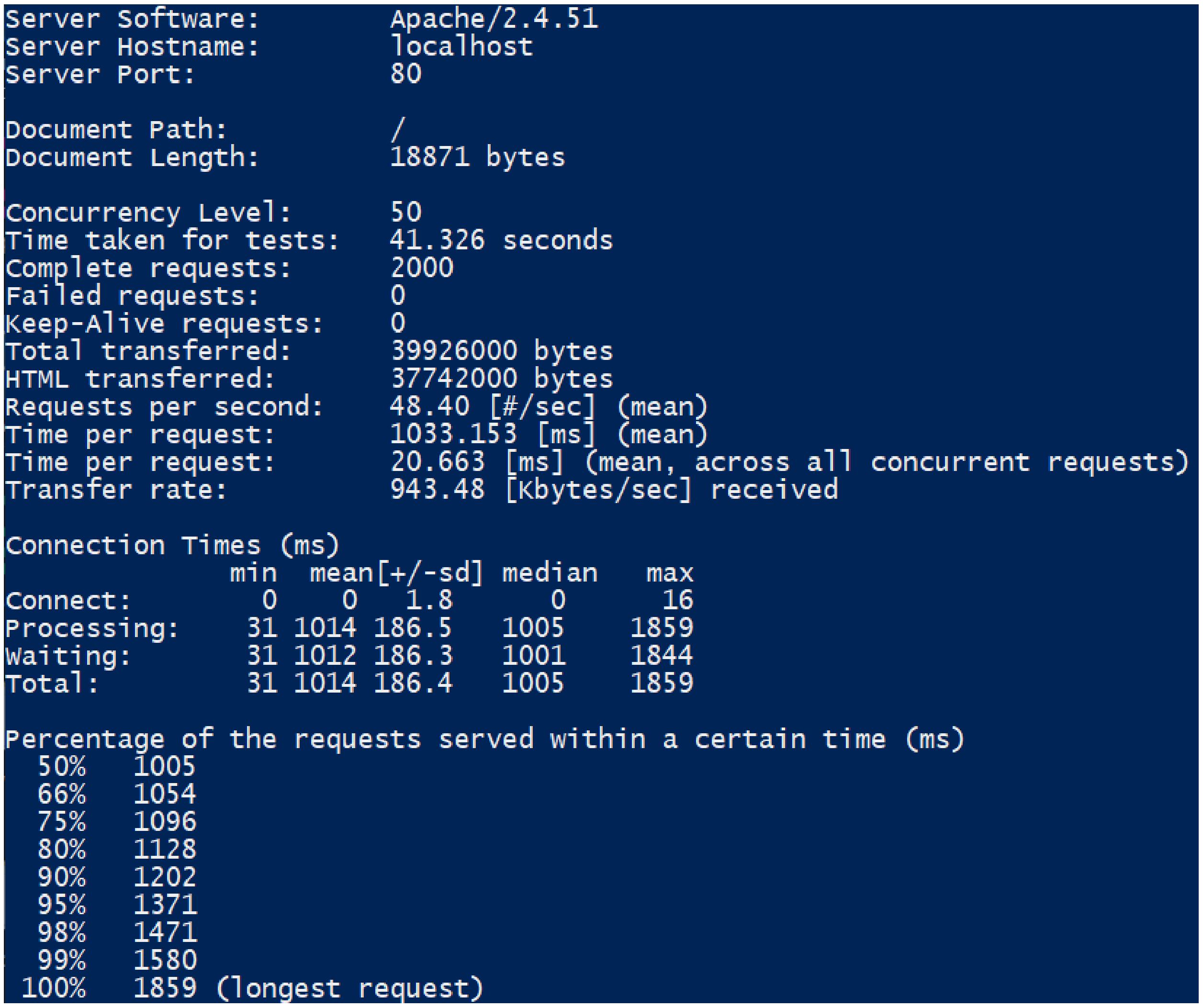


```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        18871 bytes

Concurrency Level:      10
Time taken for tests:   18.297 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      19963000 bytes
HTML transferred:       18871000 bytes
Requests per second:    54.65 [#/sec] (mean)
Time per request:       182.966 [ms] (mean)
Time per request:       18.297 [ms] (mean, across all concurrent requests)
Transfer rate:          1065.51 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.8      0      16
Processing:    31  181  38.3    172     453
Waiting:       31  178  38.0    172     453
Total:         31  181  38.3    172     453

Percentage of the requests served within a certain time (ms)
  50%    172
  66%    188
  75%    203
  80%    203
  90%    230
  95%    250
  98%    285
  99%    307
 100%    453 (longest request)
```

Figure 15 - Home Page Test 1

As we can see there were no failed requests, and the longest request was 453 milliseconds, which would be considered acceptable.

Since the server was able to handle that amount of load, we increased it to 2000 total requests with 50 concurrent requests again on the home page.



```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        18871 bytes

Concurrency Level:      50
Time taken for tests:   41.326 seconds
Complete requests:      2000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      39926000 bytes
HTML transferred:       37742000 bytes
Requests per second:    48.40 [#/sec] (mean)
Time per request:       1033.153 [ms] (mean)
Time per request:       20.663 [ms] (mean, across all concurrent requests)
Transfer rate:          943.48 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.8      0      16
Processing:    31 1014 186.5   1005    1859
Waiting:       31 1012 186.3   1001    1844
Total:         31 1014 186.4   1005    1859

Percentage of the requests served within a certain time (ms)
  50%   1005
  66%   1054
  75%   1096
  80%   1128
  90%   1202
  95%   1371
  98%   1471
  99%   1580
 100%   1859 (longest request)
```

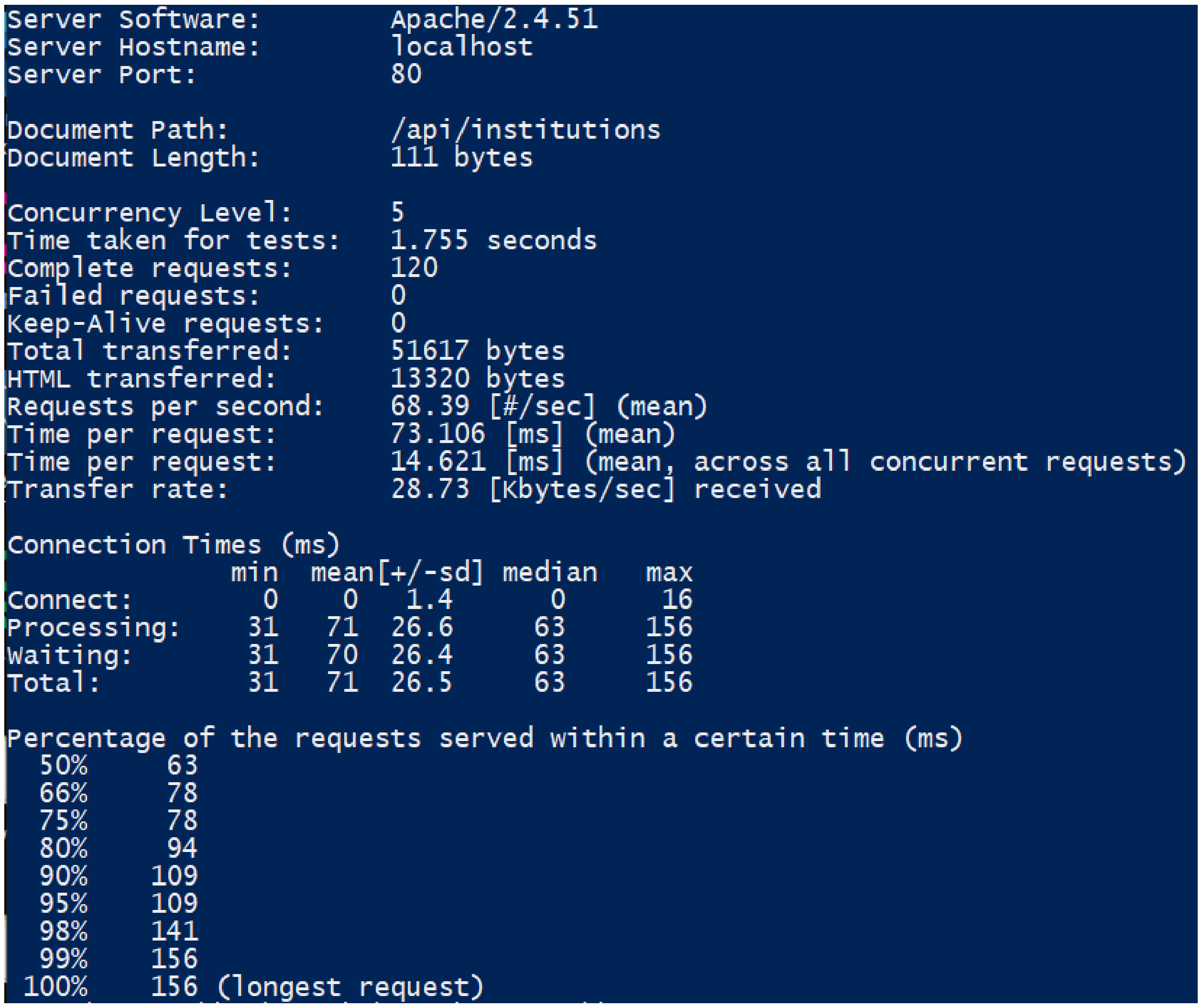
Figure 16 - Home Page Test 2

With 50 concurrent requests and 2000 total requests, most requests took longer than one second. While this is not ideal, this would be considered an unusual amount of load for this web application, and no requests failed.

5.8.2 API - Get Institutions

The next test we ran was on our API method in which we receive the list of institutions from workdaydb. This test is important because every time the Major and Courses page is loaded, this API call is run asynchronously.

The first test was run with 120 total requests and five concurrent requests.

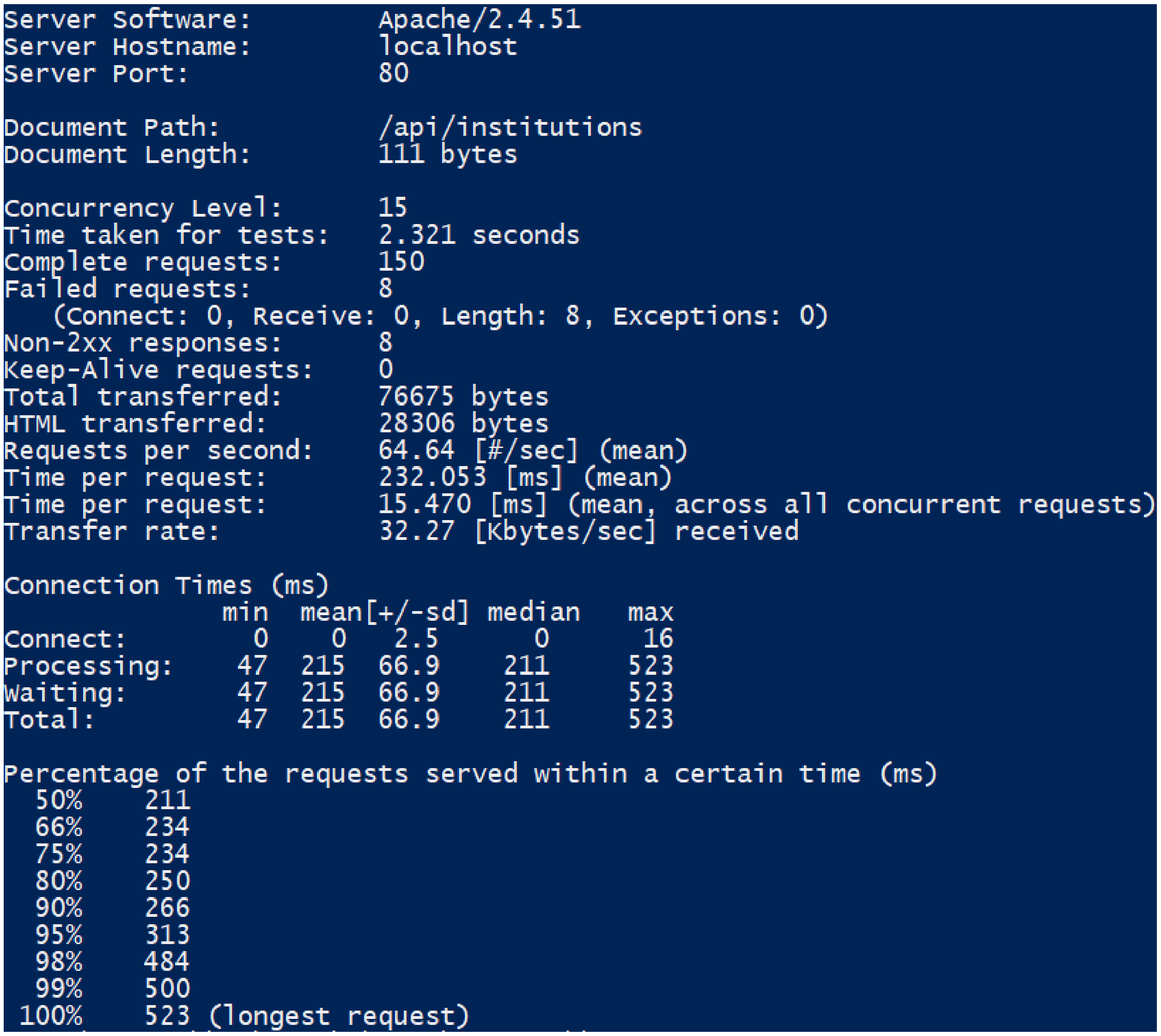


```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /api/institutions
Document Length:        111 bytes

Concurrency Level:      5
Time taken for tests:   1.755 seconds
Complete requests:      120
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      51617 bytes
HTML transferred:       13320 bytes
Requests per second:    68.39 [#/sec] (mean)
Time per request:       73.106 [ms] (mean)
Time per request:       14.621 [ms] (mean, across all concurrent requests)
Transfer rate:          28.73 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.4      0      16
Processing:    31   71  26.6     63     156
Waiting:       31   70  26.4     63     156
Total:         31   71  26.5     63     156

Percentage of the requests served within a certain time (ms)
  50%     63
  66%     78
  75%     78
  80%     94
  90%    109
  95%    109
  98%    141
  99%    156
 100%    156 (longest request)
```

Figure 17 - Get Institutions Test 1

The results of this test were good, as all requests were completed within 156 milliseconds. By the time this request is received, even in the worst case, it is unlikely that the prospective student user would notice a delay in being able to select an institution.

The second test was run with 150 total requests and 15 concurrent requests.



```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /api/institutions
Document Length:        111 bytes

Concurrency Level:      15
Time taken for tests:   2.321 seconds
Complete requests:      150
Failed requests:        8
   (Connect: 0, Receive: 0, Length: 8, Exceptions: 0)
Non-2xx responses:      8
Keep-Alive requests:    0
Total transferred:      76675 bytes
HTML transferred:       28306 bytes
Requests per second:    64.64 [#/sec] (mean)
Time per request:       232.053 [ms] (mean)
Time per request:       15.470 [ms] (mean, across all concurrent requests)
Transfer rate:          32.27 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   2.5      0      16
Processing:    47  215  66.9    211     523
Waiting:       47  215  66.9    211     523
Total:         47  215  66.9    211     523

Percentage of the requests served within a certain time (ms)
  50%    211
  66%    234
  75%    234
  80%    250
  90%    266
  95%    313
  98%    484
  99%    500
 100%    523 (longest request)
```

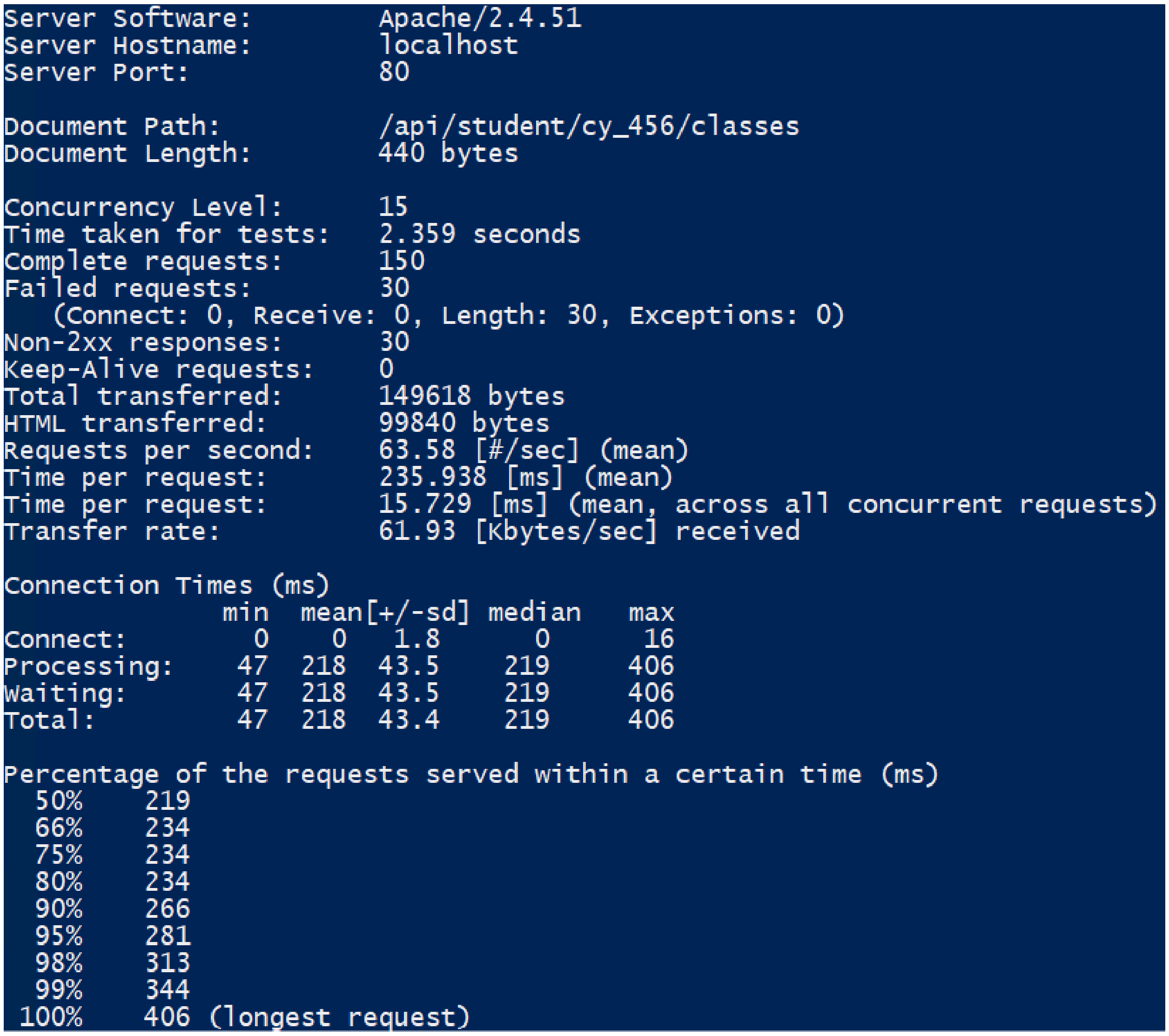
Figure 18 - Get Institutions Test 2

Even with 15 requests at a time, the server performed well until the 90th percentile of requests.

5.8.3 API - Get Saved Classes

Just like the previous request runs asynchronously on load, so does the Get Saved Classes request. This test was run with four saved classes, which would be typical for a transfer student.

Get Saved Classes with 15 concurrent requests, 150 total requests:



```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /api/student/cy_456/classes
Document Length:        440 bytes

Concurrency Level:      15
Time taken for tests:   2.359 seconds
Complete requests:      150
Failed requests:        30
   (Connect: 0, Receive: 0, Length: 30, Exceptions: 0)
Non-2xx responses:      30
Keep-Alive requests:    0
Total transferred:      149618 bytes
HTML transferred:       99840 bytes
Requests per second:    63.58 [#/sec] (mean)
Time per request:       235.938 [ms] (mean)
Time per request:       15.729 [ms] (mean, across all concurrent requests)
Transfer rate:          61.93 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.8      0      16
Processing:    47  218  43.5    219     406
Waiting:       47  218  43.5    219     406
Total:         47  218  43.4    219     406

Percentage of the requests served within a certain time (ms)
  50%    219
  66%    234
  75%    234
  80%    234
  90%    266
  95%    281
  98%    313
  99%    344
 100%    406 (longest request)
```

Figure 19 - Get Saved Classes Test

Despite querying both workdaydb and middlewaredb, this API call performed well. Especially since we have a spinner to indicate loading while courses are loading, users shouldn't mind waiting in almost all cases less than three-tenths of a second.

5.8.4 Four-Year Plan Table Page

The last test was a test of the Four-Year Plan Table page, which displays the result of the four-year plan query in a table format. Classes that successfully transfer to Iowa State are crossed-off.

The first test had 15 concurrent requests and 150 total requests:

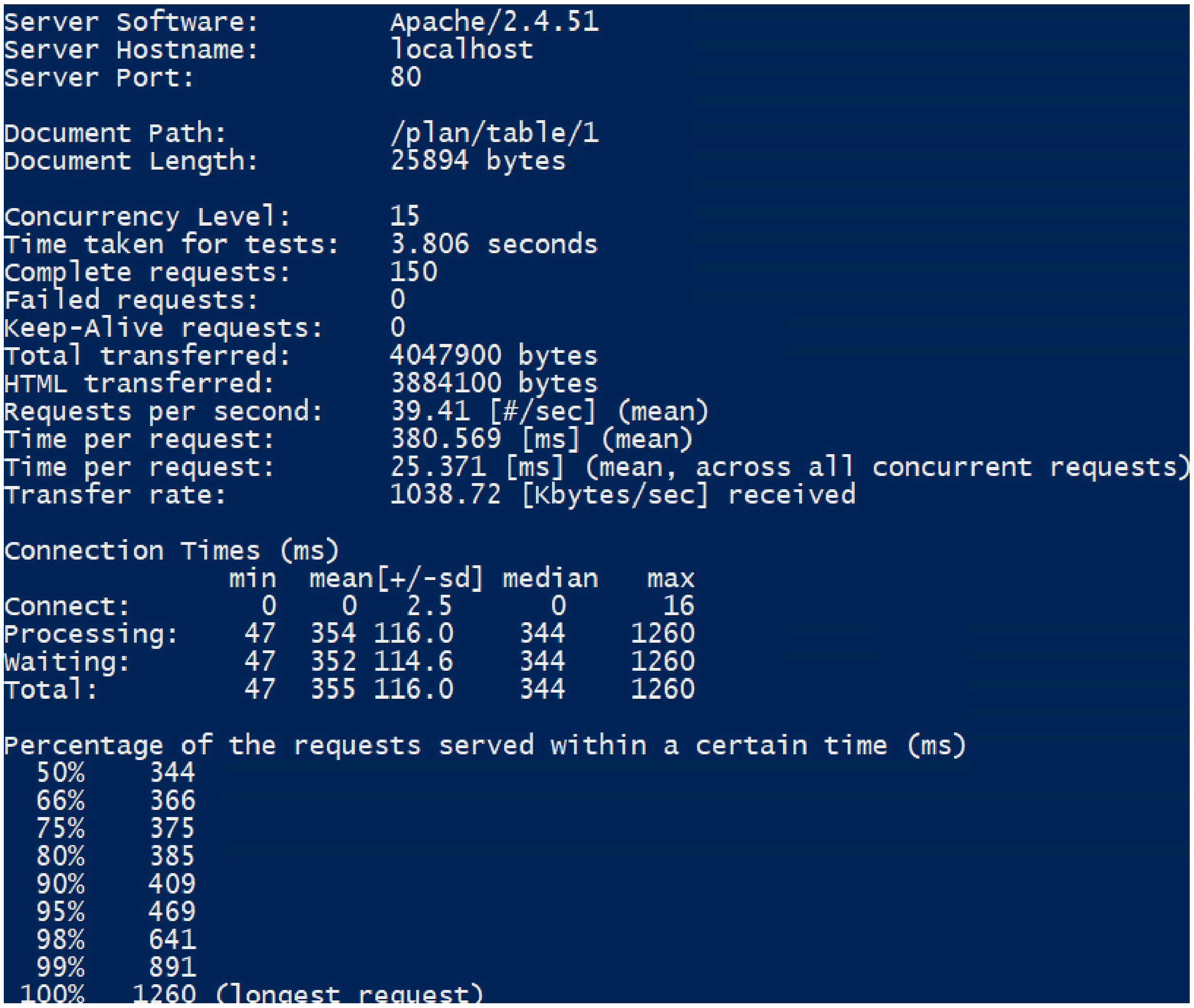
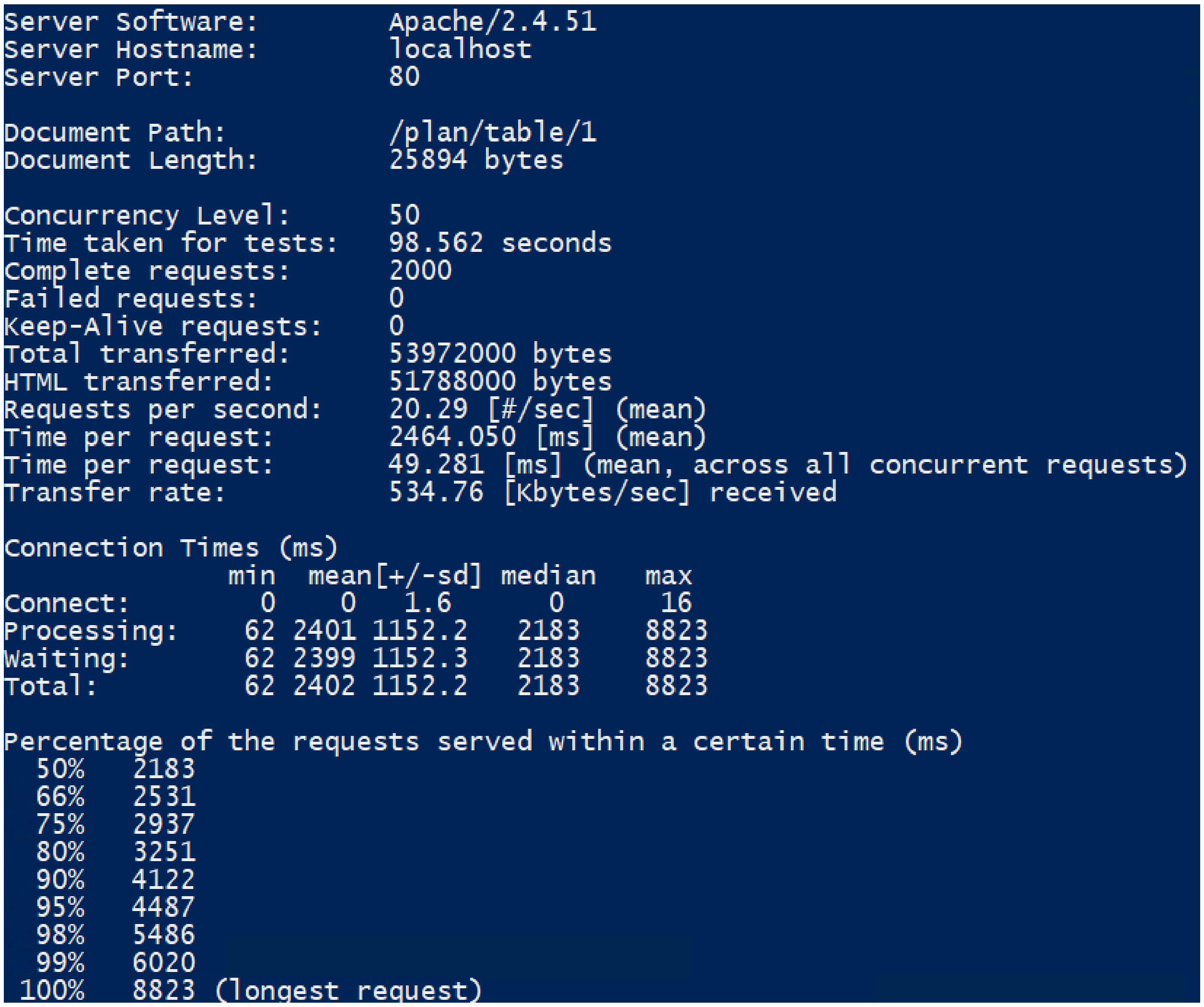

```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /plan/table/1
Document Length:        25894 bytes

Concurrency Level:      15
Time taken for tests:   3.806 seconds
Complete requests:      150
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      4047900 bytes
HTML transferred:       3884100 bytes
Requests per second:    39.41 [#/sec] (mean)
Time per request:       380.569 [ms] (mean)
Time per request:       25.371 [ms] (mean, across all concurrent requests)
Transfer rate:          1038.72 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   2.5      0      16
Processing:    47  354 116.0    344    1260
Waiting:       47  352 114.6    344    1260
Total:         47  355 116.0    344    1260

Percentage of the requests served within a certain time (ms)
  50%    344
  66%    366
  75%    375
  80%    385
  90%    409
  95%    469
  98%    641
  99%    891
 100%   1260 (longest request)
```

Figure 20 - Four-Year Plan Table Page Test 1

This test performed well, as only two percent of requests took over 6 tenths of a second.

Attempting to establish maximum load, a test was run with 2000 total requests and 50 concurrent requests again on the table page.



```
Server Software:        Apache/2.4.51
Server Hostname:        localhost
Server Port:            80

Document Path:          /plan/table/1
Document Length:        25894 bytes

Concurrency Level:      50
Time taken for tests:   98.562 seconds
Complete requests:      2000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      53972000 bytes
HTML transferred:       51788000 bytes
Requests per second:    20.29 [#/sec] (mean)
Time per request:       2464.050 [ms] (mean)
Time per request:       49.281 [ms] (mean, across all concurrent requests)
Transfer rate:          534.76 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.6      0      16
Processing:    62 2401 1152.2   2183    8823
Waiting:       62 2399 1152.3   2183    8823
Total:         62 2402 1152.2   2183    8823

Percentage of the requests served within a certain time (ms)
  50%   2183
  66%   2531
  75%   2937
  80%   3251
  90%   4122
  95%   4487
  98%   5486
  99%   6020
 100%   8823 (longest request)
```

Figure 21 - Four-Year Plan Table Page Test 2

The load on the server proved to be too much for this test as most requests took over two seconds.

5.8.5 Conclusion

- Our web application performed well overall under load tests using ApacheBench.
- Selected API calls such as Get Saved Classes and Get Institutions performed well even with 150 requests in the span of fewer than three seconds.
- For both our home page and especially the Four-Year Plan Table page, 2000 total requests with 50 concurrent users were too much for the system.
- We estimate that up to 40 active concurrent student users would be able to utilize the application without noticing performance issues.
- The maximum load could be increased by getting better web server hardware and/or by using a load balancer.

5.9 Results

Our extensive testing strategy resulted in a satisfactory application that our client found acceptable. Unit testing ensured that our client-side validation methods and server-side adapted models functioned properly on their own. Interface testing ensured correctness of our API and web routes. Integration testing ensured that both of our databases, models, and the four-year plan calculation feature worked even when interacting with a large number of different components. System testing allowed us to verify the whole stack of our application from the perspective of a user. Manual testing also assisted us on an as-needed basis and for features that did not have test coverage. We were able to measure the performance of the system using both manual and load-testing. Finally, we used GitLab CI/CD to continuously run our automated tests to prevent regressions. A total of 488 pipelines ran over the course of our project, of which, 291 were successful. 126 totals tests were written, all of which ran in the pipeline. 24 of these were system tests, 20 were interface tests, 25 were integration tests, and 57 were unit tests.

The diagram below outlines the flow of implementing a feature and the responsibilities of each team member in the process.

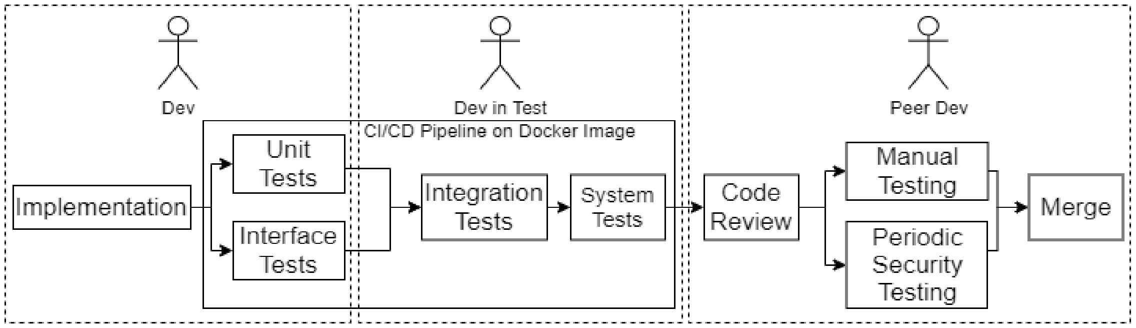


Figure 22 – Testing Flow Diagram

Requirement/Constraint	How it is addressed
Mobile-friendly project	System/Functional Testing
Use the Iowa State website template	Unit and System/Functional Testing
Account creation for prospective students has the format of an admissions prospect record	Unit and System/Functional Testing
Able to interact with a workday backend, or a mocked version	Interface and System/Functional Testing

Accept as input transfer courses and grades from other universities	Unit, Interface, Integration, and System/Functional Testing
Use the inputted transfer courses to determine transferability at ISU	Unit, Interface, Integration, and System/Functional Testing
Output a four-year plan of the intended major of the prospective student in table format based on an existing four-year plan	Unit, Interface, Integration, and System/Functional Testing
Output a four-year plan of the intended major of the prospective student in flowchart format based on an existing four-year plan	Unit, Interface, Integration, and System/Functional Testing
Both views (flowchart and table) display courses crossed off as well as the transferred course name	System/Functional Testing
Able to download a PDF file in flowchart and table format	Manual Testing
Exported PDF files display creation date	Manual Testing
Premade accounts exist for administrators and prospective students	Interface and System/Functional Testing
Transfer courses can be linked to a prospective student account	Interface and System/Functional Testing
Linked data can be changed and deleted	Interface and System/Functional Testing
Guest prospective student users can use application without account	System/Functional Testing

Unofficial evaluation disclaimer	System/Functional Testing
Display other resources links	System/Functional Testing
Allow selection of preferred major	Manual Testing
Allow administrator users to view aggregate data regarding individual prospective student users	Interface and System/Functional Testing
Allow administrator users to download individual student data in an Excel-accessible format	Interface and System/Functional Testing
Allow administrator users to view aggregate data regarding prospective student users	Manual Testing
Allow administrator users to reach out via email to prospective student users	Interface and System/Functional Testing
Respond within 1 second to a four-year plan query	Load and Manual Testing
User is able to complete a session in under ten minutes	Acceptance Testing
User interface shall be easy to navigate	Acceptance Testing

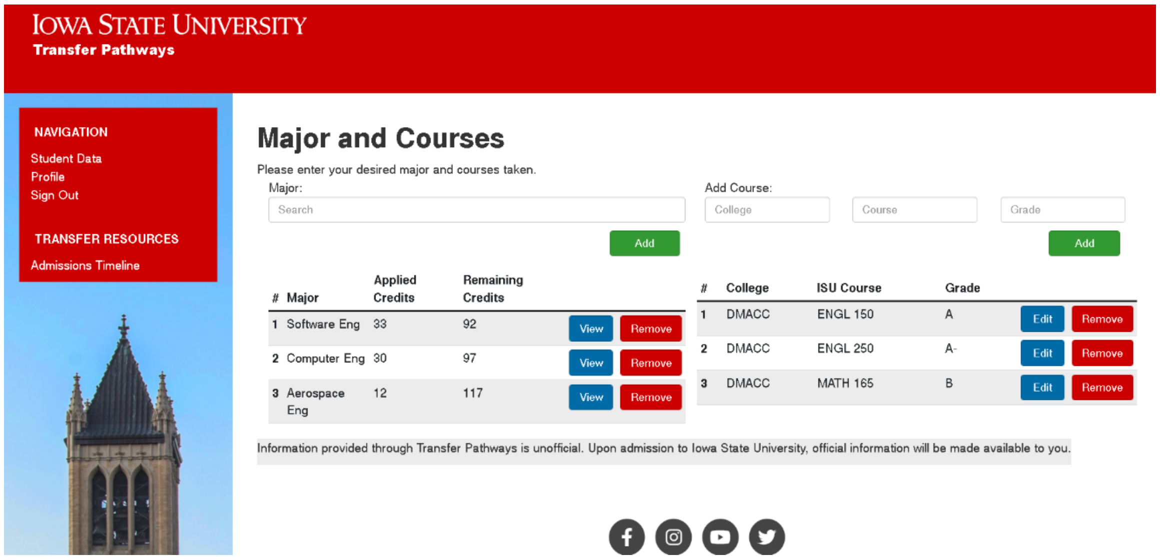
Table 5 – How Testing Addresses Requirements/Constraints

6. Implementation

This section goes over the basic implementation details of all aspects of our project. It should help the reader understand how each major component works.

6.1 Prototype Implementation

One major early risk to our project was that the page design and screen flow would lead to a poor user experience. We recognized the severity and probability of such a scenario and completed a prototyping phase in the first semester. We began to iterate on possible UI designs, taking inspiration from similar products while improving them with our own ideas. We took these ideas to our client who gave us feedback on them. Eventually, we reached a UI that everyone could agree on and created a prototype of it as seen below. The prototype is a plain HTML website with only navigation functionality. The HTML prototype served as the foundation for the rest of the frontend implementation.



Major and Courses

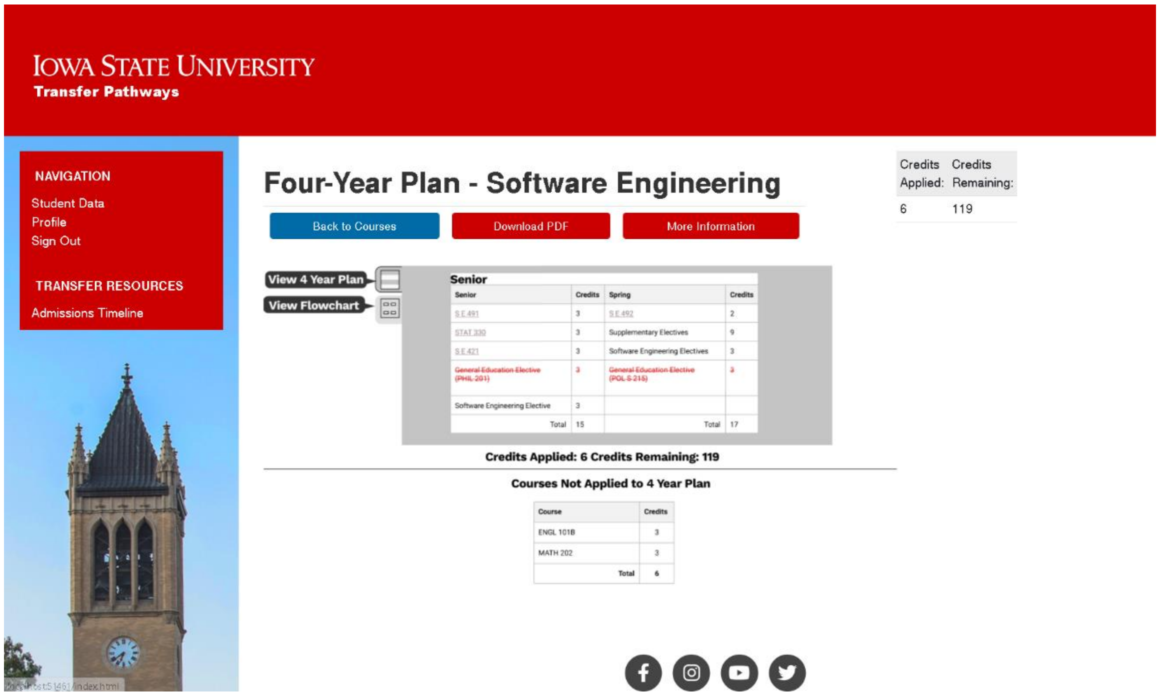
Please enter your desired major and courses taken.

#	Major	Applied Credits	Remaining Credits		
1	Software Eng	33	92	View	Remove
2	Computer Eng	30	97	View	Remove
3	Aerospace Eng	12	117	View	Remove

#	College	ISU Course	Grade		
1	DMACC	ENGL 150	A	Edit	Remove
2	DMACC	ENGL 250	A-	Edit	Remove
3	DMACC	MATH 165	B	Edit	Remove

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Figure 23 - Major and Courses Prototype Page



Four-Year Plan - Software Engineering

Credits Applied:	Credits Remaining:
6	119

Senior

Senior	Credits	Spring	Credits
S E 491	3	S E 492	2
STAT 330	3	Supplementary Electives	9
S E 421	3	Software Engineering Electives	3
~~General Education Elective (PHIL 201)~~	~~3~~	~~General Education Elective (POL S 215)~~	~~3~~
Software Engineering Elective	3		
Total	15	Total	17

Credits Applied: 6 Credits Remaining: 119

Courses Not Applied to 4 Year Plan

Course	Credits
ENGL 101B	3
MATH 202	3
Total	6

Figure 24 - Four-Year Plan Table Prototype Page

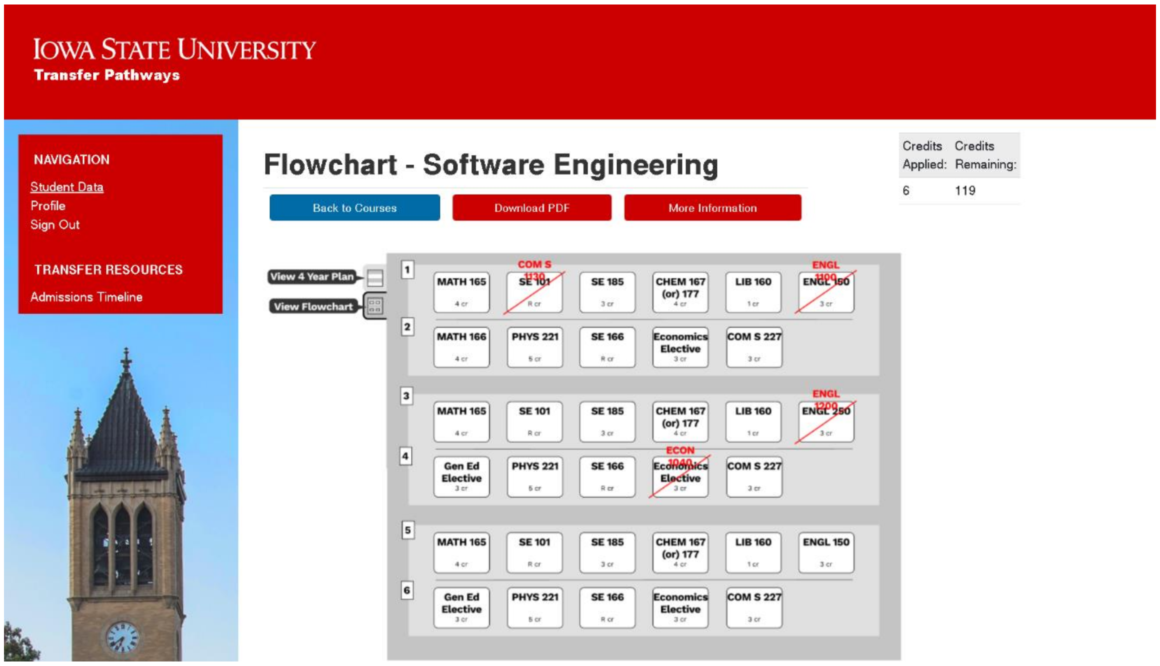


Figure 25 - Four-Year Plan Flowchart Prototype Page

Prototype shown above available at [Transfer Pathways Demo • Iowa State University (iastate.edu)]()

6.2 Frontend Implementation

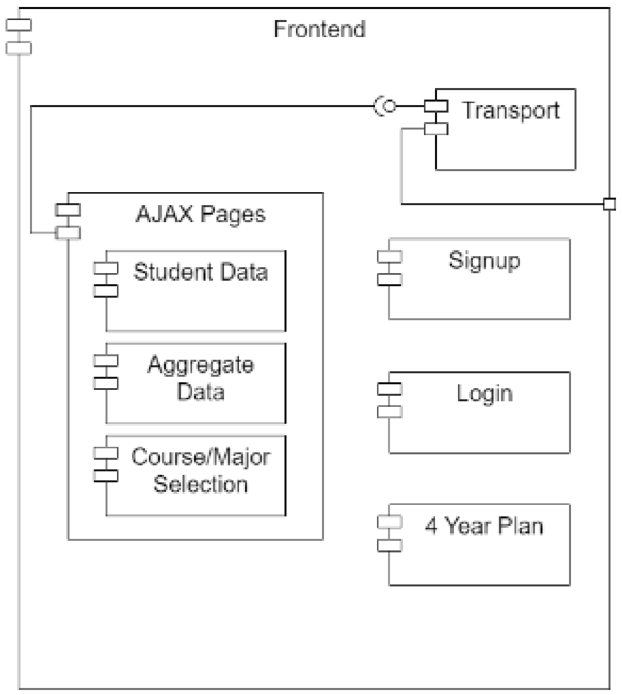


Figure 26 - Frontend Component Diagram

The frontend can be categorized into two parts: the Laravel views and the AJAX requests. The Laravel views are essentially the server-side rendering of the HTML documents, written in PHP. The AJAX requests are the client-side JavaScript code used for dynamically loading data onto a live page, rather than requesting the server to render a new Laravel view.

The majority of the pages are purely Laravel views with no AJAX code. In Laravel, views are able to extend other views. Each page extends a base Iowa State theme page for a consistent user experience and a modular software design. Each page has an associated controller that handles feeding pertinent data to the view for rendering. One example of this is the Plan Controller. It is responsible for picking either the flowchart view or the table view based on the end user's request. After deciding which view to render, it fetches the four-year-plan from the data providers and sends it to the view for rendering.

The main use of AJAX can be found on the Enter Courses/Majors page. All other uses of AJAX follow a similar pattern. Since the user will constantly be adding new courses and majors, it makes the most sense to utilize AJAX requests here. It reduces the server workload and reduces the latency between a user action and the user seeing its effect. After the user adds, deletes, or edits a course/major, the corresponding function in the FrontendTransport class will be called. This function asynchronously makes a request to the middleware and returns a JavaScript promise. When that promise is resolved, the consumer can update the page to reflect the updated state of the server (e.g., add a new row for the newly added course).

6.3 Middleware Implementation

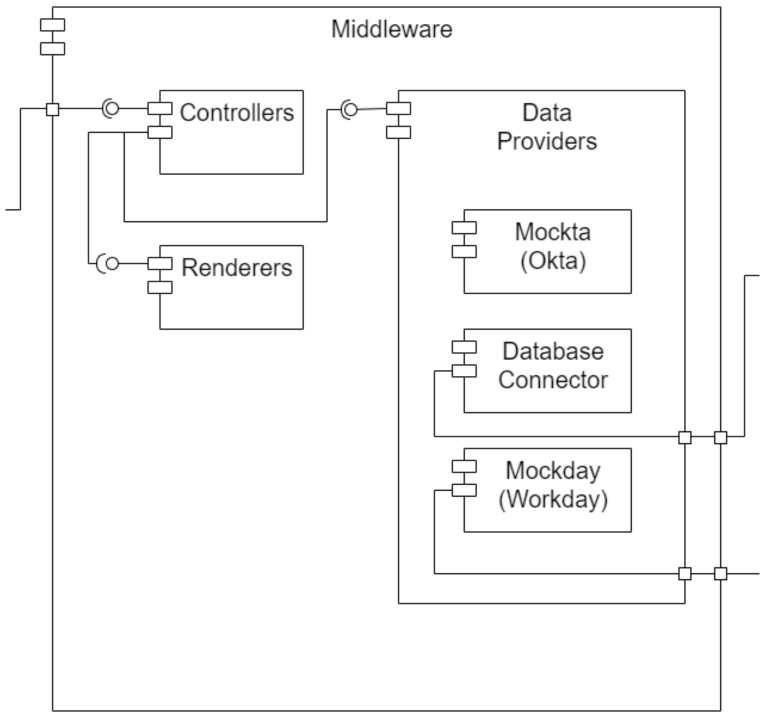


Figure 27 - Middleware Component Diagram

The implementation of the middleware can be seen by the component diagram above. Requests come into the middleware through the controllers component. This component consists of the controllers that are called based on a certain request route. These controllers will first verify that the user is authorized to make that request based on their session information. Then, they will get or set the appropriate data in the data providers. After the data processing is complete, the controller will call a function in the renderers to create the appropriate view with the processed data to send back to the frontend. This view will then display on the user's screen.

The data providers component is further broken down into three components. These components are Mockta, the database connector and Mockday. The database connector component is used to communicate with the permanent database of the system. It performs queries to either get, add, delete or edit data from the database. The Mockday component is our temporary implementation of Workday. This component does everything Workday will eventually need to do by querying the Mockday database and manipulating the data stored in there. The Mockta component is a self-contained component that is our temporary implementation of Okta. This component does everything Okta will eventually need to do.

6.4 Database Implementation

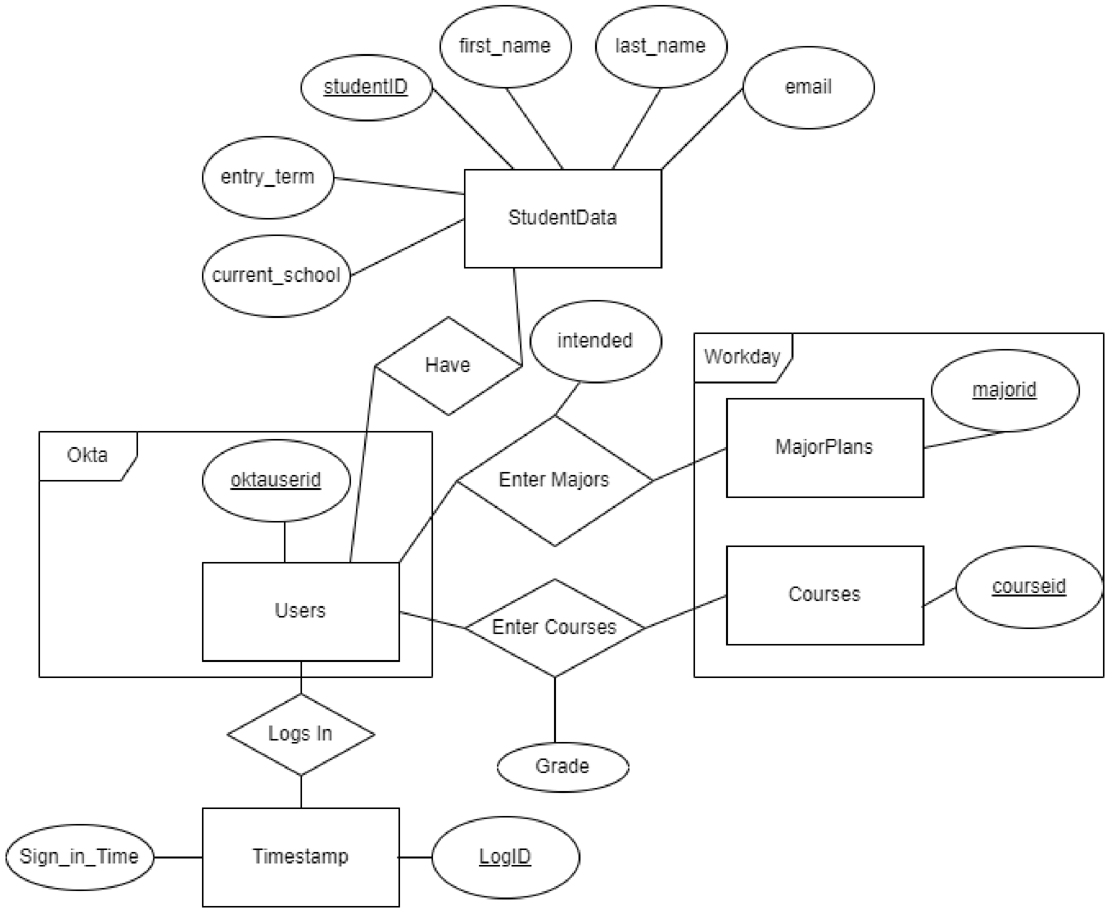


Figure 28 - middlewaredb ER Diagram

Although much of our data is retrieved from either Workday or Okta, we do still have a need to store user data in a MySQL database located on our server. How the database is set up can be seen in the ER diagram above. The boxes labeled "Okta" and "Workday" are not actually a part of the database. They are shown there to display how the table data is linked to outside information from Okta and Workday. For example, an okta user id is stored in the enter majors table with a major id from Workday. This shows that a specific user with a specific okta id has entered the specific major as a major they are interested in. The database also holds data about what courses students have taken, when students have logged in and data about each student that has registered. All queries to this database are handled in the DatabaseConnector class. This class implements a DatabaseConnectorInterface that has very detailed explanations for what each method does, what the parameters are and the expected output. This allows future developers to easily switch out this database with a new one if they wanted to use a NoSQL database or change something else.

6.5 Okta Implementation

We were not able to get access to Iowa State University's implementation of Okta for a number of different reasons, so we needed to make our own mocked version of Okta for authentication purposes. We looked at the API for the Iowa State implementation of Okta and noticed that it had the ability to authenticate different users and add different users. We took advantage of this in our project and created a couple of well documented methods that authenticate a set of hardcoded users and create a unique okta id for a new user. These methods are in an OktaConnector class. This class implements the OktaConnectorInterface that has very detailed explanations for what each method does, what the parameters are and the expected output. This allows Iowa State IT to only have to implement their own version of the OktaConnector class to successfully integrate with Okta.

6.6 Workday Implementation

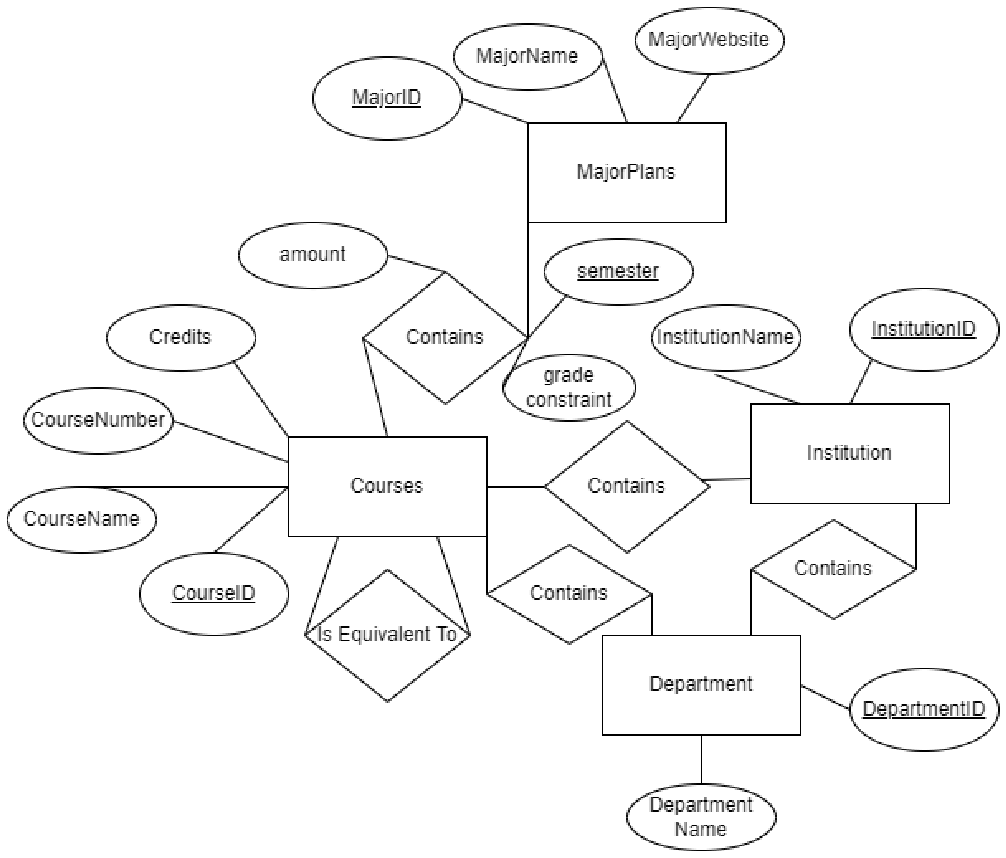


Figure 29 - workdaydb ER Diagram

The Iowa State implementation of Workday is still a couple years away from being completed so we utilized our contacts in Iowa State IT to understand what the service will provide. Similarly to our Okta implementation, we wanted to have a set of hardcoded data that would resemble what Workday would really give us in the future. We decided to make a temporary Workday MySQL database to hold this information. The implementation of this database is shown in the ER diagram above. The data in this database is used for different recognized institutions, departments for those institutions, courses and four-year plans for Iowa State majors. Like the OktaConnector, we created a class called WorkdayConnector that is used to connect to necessary Workday data. In this case we complete database queries for necessary information. This class also implements an interface that has detailed explanations for what each method does, what the parameters are and the expected output. All Iowa State must do to connect to Workday is implement their own version of the WorkdayConnector class that calls the necessary Workday API endpoints and make sure the input and output of the functions are what we defined in the interface, WorkdayConnectorInterface.

6.7 WAMP Server Implementation

From early on in the development phase of our project, we have utilized a WAMP server to run the production environment of our application. A WAMP server was chosen because it bundles Apache, MySQL, and PHP into one installation, making the server easy to maintain (the W stands for Windows, as our server is a Windows server). Upgrades to the individual services can be installed from the WAMP server website if needed. After installing the WAMP server, only minor modifications were needed to run the website. First, the directory structure was modified for auto deployment. Next, port 80 was opened in the firewall to allow incoming HTTP traffic. Then, the Apache http-vhosts.conf file was modified to support our directory structure and allow all incoming traffic. Finally, database credentials were set up using MySQL Workbench, which does not come with the WAMP server. Using the WAMP server has made running our web application very easy with little maintenance required since the initial setup.

Additionally, auto deployment was set up to deploy the most recent version of the Transfer Pathways main branch to the server, assuming the first two testing stages of the pipeline were successful. The auto deployment uses a shell (Powershell) GitLab Runner to execute commands on the server and deploy the website. The GitLab Runner will move the previous version of Transfer Pathways to the Backup directory and the Deploy directory will be empty when finished. Default contains the most recent version of Transfer Pathways, which the server runs at all times. Auto deployment means that one button click on GitLab can deploy the project to our server in about a minute.

7. Professionalism

Professionalism was of utmost importance to our project, and we have looked over the ACM SE Ethics and Professional Practices and thought about how our project relates to this code.

7.1 Areas of Responsibility

Area of Responsibility	Definition	IEEE/ACM SE Ethics and Professional Practices
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence	Product. Software engineers shall ensure that their products and related modifications meet the highest professional standards possible
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs	Client and employer. Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest
Communication	Report work truthfully, without	Judgment. Software engineers shall

Honesty	deception, and understandable to stakeholders	maintain integrity and independence in their professional judgment
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders	Client and employer. Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest
Property Ownership	Respect property, ideas, and information of clients and others	Client and employer. Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest
Sustainability	Protect the environment and natural resources locally and globally	Public. Software engineers shall act consistently with the public interest
Social Responsibility	Produce products and services that benefit society and communities	Public. Software engineers shall act consistently with the public interest

Table 6 – Areas of Responsibility

Work Competence

One aspect of the SE code of ethics is product. Simply put, this code requires that engineers focus on building high quality products. Engineers should strive to understand all the specifications of the software they are developing. They should take into account factors from every angle, including environmental, cultural, legal, economic and ethical aspects. They should perform adequate amounts of testing and debugging to ensure they can deliver a high quality, fully functioning product to the client, customer, or employer. This code is very similar to the NSPE version at its base. The main idea of both is to strive for high quality. The SE version is specifically focused on software development, and makes points regarding testing, debugging and other aspects specific to software engineering.

Financial Responsibility

The client and employer codes in the SE code of ethics talk about having your client's or employer's best interests in mind, without going against the public interest. This means staying faithful to who you are working for. Keep confidential information known to only those involved and report mishandling of information or software. Always stay honest when communicating progress, expectations, limitations, and abilities. All focus should be on the client's needs and benefits, within the scope of public interest. This code differs from the NSPE version, as it does not specifically use the word financial or any like term. It rather focuses on integrity to the client and their interest.

Communication Honesty

The parallel principle to Communication Honesty in the SE Code of Ethics is Judgment. Engineers have a responsibility to communicate honestly and openly with coworkers, clients, and other stakeholders. They must engage in all evaluation objectively regardless of the consequences of an unfavorable conclusion. This principle differs from the NSPE version by focusing more on general objectivity within the organization as opposed to communicating with the public.

Health, Safety, Well-Being

The principle most similar to Health, Safety, and Well-Being in the SE Code of Ethics is Client and Employer. Engineers should be beholden to issues of social concern. They are responsible for reporting and acting on known ethical issues. Depending on the product, this could involve sensitive personal information, vital medical data, or plenty of other at-risk information. This could manifest as emphasizing data security, or, on a more sinister note, ensuring that this data is not being sold by the company. This principle differs significantly from the NSPE version because in software, health, safety, and well-being do not have as much of a direct effect on decision making, excluding the examples given above.

Property Ownership

The client and employer codes go along best with the idea of property ownership. This area of the SE code of ethics stresses the importance of keeping your client/employer's best interests in mind. This means the information of the client and software being developed is kept confidential. Any communication should be kept honest and non-deceptive, so the client/employer is always informed accurately. Any software and physical technologies being used in development should be used and obtained with proper consent and knowledge from the client. These codes are very similar to the NSPE ideas of acting as a faithful agent to the client.

Sustainability

The ACM code of ethics discusses the importance of designing and developing systems for a safe natural environment and the potential damage it can have to either the local or global environment. This code of ethics was very similar to the NSPE version. However, the NSPE code did include one difference, it provided a reason for sustainability which was to protect the

environment for future generations. This was not directly mentioned in the ACM code of ethics, therefore, it seemed as though it was more so focused on the present impact it has on the environment.

Social Responsibility

The public code in the SE code of ethics is the same as the NSPE code for social responsibility. The SE ethics talk about always keeping the general public in mind when building software. While an engineer should always focus on their clients/employer's best interest, they must do so in the scope of society. They must not build anything that would diminish the quality of life, privacy, or the environment of the affected society. All publicly released information, testing, and documentation should be honest and understandable. There should be no shortcuts taken when testing or approving different technologies that may have an impact on the society. Engineers should take full responsibility in developing high quality software that will benefit their client without the expense of anyone or anything else.

7.2 Project Specific Professional Responsibility Areas

Area of Responsibility	**Application**	**Performance**
Work Competence	Yes, it was necessary for our team members to be competent with the work required to implement the project. The skills and abilities that were applied to the project followed the same technology standards ISU IT has in place such as PHP, Bootstrap, and HTML. We also had a need for a clear idea of what is expected in terms of the user experience and flow according to our client.	High We were able to gain insight not only from ISU IT but our client as well to understand what was expected of us from the project and its requirements. Although it may have taken more meetings and emails than first thought, we formulated a clear path and direction to follow. Our team familiarized ourselves with the technology skills that were required to develop this application such as PHP. Previously, the majority of our group members did not have any experience with it.
Financial Responsibility	Yes, our group needs to deliver our Transfer Pathways Tool in a way that provides a high degree of value to the Admissions Department at a low cost.	Medium In the first semester, we presented our client with a UI prototype that satisfied their needs. Since then, we have developed a program that follows the prototype, and implements all of the features of the UI, middleware, and backend. However, we cannot fully analyze the value of our project until ISU IT implements it with their backend services in the near future. For cost, we can say that we are performing high, as all that we will require cost-wise is server space.

Communication Honesty	Yes, communication honesty applied to our project because we had multiple stakeholders who expected a good product. Honestly communicating to our stakeholders was an ethical practice in our case, so that our stakeholders had an accurate idea of project progress.	High From very early on in the project, we maintained a high degree of communication with all stakeholders, including our faculty advisor and our two main contacts from the Admissions Department. We met frequently with all stakeholders via video chat in addition to email when feasible. Finally, we believe ourselves to have been honest about the progress of our work to all stakeholders. We kept them updated on major issues we had come across and any changes that had to be made from the original design.
Health, Safety, Well-Being	Yes, this applied to our project; while no one can be physically harmed from the website, psychological harm could be done. If users receive incorrect or misleading four-year plans, they can be hurt by incorrect planning and preparations.	High We have included multiple disclaimers in our UI in order to make sure transfer students understand this information is not final.
Property Ownership	Yes, one item that is of concern to a user is the ownership of the data they give to our system. Additionally, we had concerns of ownership issues with the course information from other schools and institutions. We were also wary of using ISU-owned materials like the Workday API, the ISU web templates, and the Okta API - being sure to credit them in our final product, even when mocked in our implementation.	Medium Any user that submits information to our tool to find their modified four-year plan has their information accessible by admin users from the department of admissions. For issues with ISU owned materials, we were able to get access to ISU themes and web templates for the design of the UI. We did not get access to the Workday API yet as ISU IT has not completed development. We also did not get access to ISU's Okta API for security purposes. Instead, we have created our owned mocked authentication methods. ISU's Okta will have to be linked with our project later on when they finish up and implement it with the Workday API and backend.

Sustainability	No, since our project is updating a website using a server that already exists it has no environmental impact.	N/A
Social Responsibility	Yes, this product can benefit society in many ways. It allows transfer students to more easily be able to plan their course work at Iowa State. It also allows administrators to learn what students are generally interested in while attending Iowa State. All of this was achieved using an easy-to-understand user interface that any person should be comfortable with.	High We believe we performed highly in this category. We did a lot of research and iterated using feedback on a prototype of our website to make it as user friendly as possible. Making it easy to use allowed it to appeal to the most people. This allows the greatest number of individuals in society to benefit.

Table 7 – Project Specific Responsibility Areas

7.3 Most Applicable Professional Responsibility Area

Communication honesty was crucial to our project for two main reasons. First, the project is a public, outward facing representation of ISU to prospective students. For this reason, we needed to have consistent, open, and honest communication with ISU Admissions to ensure we accurately portrayed the image ISU wishes to project. Additionally, we wanted the project to portray accurate and honest communication about transferring courses to prospective students as well. During planning, we had multiple meetings with our contact in ISU Admissions to discuss visual appeal, flow of information, and representation of information. These meetings also allowed us to gain approval, constructive suggestions, and feedback on our designs. Throughout the development process, we kept our client up to date with our progress. We also met with them to discuss any changes in the UI and features we were implementing.

For the second reason, the project was created with the intention of merging with a backend being created by ISU IT at a later date. While many details about such backend implementations were not determined, we needed to be open about how we made decisions on our frontend. This ensures that when ISU IT does their implementation, they know how to make the connection. Like meetings with ISU Admissions, we had multiple meetings with ISU IT as well to learn all that we could, create an understanding, and set up a consistent channel of communication while we worked on development.

8. Closing Material

The planning and prototyping done during the first semester and laid out in this document served as a comprehensive guide during the development. Throughout development, any changes made to the design or implementation were updated in this document so that this document is the most accurate description of our project.

8.1 Discussion

Our work throughout the year has resulted in what we feel is a valuable and successful project. The plan we put together in the first semester had all the resources to make the development process flow smoothly. We built each component following the design we had laid out in our plan. This included things like a blade file for the html, a page renderer and a page controller for each screen a part of our web app. The flow of these components and their data followed the diagrams we had put together in our plan.

During the development process, we had some changes to our overall design. In hindsight some of these changes could have been prevented and been a part of the original design. Many of these changes stemmed from using new tools that we weren't familiar with. We originally planned on using AJAX calls for every screen view in our program. However, after beginning development, we learned some of our pages could just be rendered without the need for AJAX. Knowing how Laravel and PHP work now, this is something we could have put in our original design. We also assumed we would get access to ISU's Okta implementation. We found out during development that we would not get this access for security reasons. We then had to decide on a new way to implement user authentication. After meeting with each of our stakeholders, we decided on mocking our own authentication methods. This design change was due to an unclear understanding of what access we would get. Creating a clear understanding of these kinds of details could have prevented design changes during development.

We also considered many non-technical factors for the development of our program. We had a set of ground rules to ensure we all communicated so everyone was on the same page, everyone was on track with their work, and we all put forth an effort to build a quality program. Our plan mentions some of the IEEE standards that we followed, as well as how we all worked in a professional and ethical manner during development.

Overall, our design document hits all the major requirements put in place by our client. We feel we kept true to our design and developed a high-quality program that has all of those required features and designs. Throughout development, we have updated this design document to be accurate to what we have developed. We had discussed all of these changes with our client through the development process, to ensure that the program would still live up to the standard we had been working towards. We believe our program and designs are properly documented and ready to be implemented with Workday when ISU IT has finished its development.

8.2 Conclusion

Our main goal throughout the last year was to develop a high-quality program that ISU Admissions could use for future potential transfer students. In order to achieve this goal, we had to begin by developing a detailed and comprehensive project plan. Throughout the development process, this plan was used to guide us through both the technical and non-technical aspects of our project. We believe we have achieved our main goal with the program we have developed.

- First, our project plan is extremely detailed, high quality and has led to successful software development.
    - We had listed all required tasks for our project and a timeline of when we would complete them. This helped ensure that our development process was as structured and efficient as possible. We used a User-Centered-Design approach while creating this plan to ensure that our designs not only met the needs of our client, but also the future users. We also made sure that all requirements have been met and all risks have been mitigated by having frequent communication with our client and ISU IT. They both approved our work on the project and were excited to see the final result.
- Secondly, we have developed many relevant skills from the project and improved on our soft skills.
    - There were multiple tools used for development that many of us had little to no experience with, especially Laravel and PHP. Building this software provided us all with experience and new skills to use in the future. Our soft skills also improved throughout the course of the last year, the biggest one being communication. We had to communicate and meet frequently with ISU IT, our client and our faculty advisor during both the planning and development phases of our project. This would be for reasons such as progress updates, discussing changes to our design, and learning information such as requirements for the software and UI. This frequent and quality communication led to our team developing the high-quality program we have made for our client.
- Third, we believe we have done a great job of following our design during development.
    - We built all of the components laid out in the plan, the functionality follows all the requirements provided by our client, and our UI closely matches the prototype that we got approved by our client. During development, we continued to use the idea of User-Centered-Design to guarantee the user experience would be beneficial and understandable. Anything that needed changing was communicated to our client or IT, and all changes were also recorded in this document, to ensure it accurately represents our current system.
- Lastly, we have ensured our system satisfies our client.
    - We have met with them and provided multiple demonstrations of our program. Our client was very pleased with what we had developed and provided us with only a few minor things that could be improved over the course of our meetings with them. We took this feedback and finished touching up some small details of the program. Our client expressed that our system will be beneficial and provide a good user experience, not only for transfer students, but also for the admissions department.

To conclude, our main goal of the year has been met. We believe we were extremely successful in developing a high quality and high value software system that will completely satisfy our client. We did so by following the detailed plan and design we had created and communicating consistently and honestly with all of our stakeholders. We are excited to see our system be fully implemented by ISU IT and we hope it provides a good and helpful experience for all the future users.

8.3 References

[1]K. Gnatek, “Engineering principles: putting our values into practice,” Medium, Jul. 30, 2020. https://medium.com/taxfix/engineering-principles-putting-our-values-into-practice-4bbc140d4fa2 (accessed Nov. 22, 2021).

[2]“Taylor Principles of Scientific Management: Meaning, Definition,” BYJU’S. https://byjus.com/commerce/taylor-principles-of-scientific-management/ (accessed Nov. 21, 2021).

[3]“Standards for Mathematical Practice | Common Core State Standards Initiative,” Common Core State Standards Initiative, 2019. http://www.corestandards.org/Math/Practice/ (accessed Nov. 21, 2021).

[4]“Iowa State University TRANSIT,” TRANSIT. https://transit.iastate.edu/ (accessed Nov. 22, 2021).

[5]“Howdy,” Howdy. https://howdy.tamu.edu/uPortal/p/tce-ui.ctf1/max/render.uP (accessed Nov. 22, 2021).

[6]“MyTransfer Credit | College Credit Transfer Check Tool | Franklin.edu,” Franklin University. https://www.franklin.edu/transferring-credit/estimate-your-transfer-credit/transfer-credit-tool (accessed Nov. 22, 2021).

8.4 Appendices

APPENDIX I - Operation Manual

Setup Instructions

These setup instructions are intended for developers and server administrators only. For other users of the system, refer to the demo instructions. Below are the local development environment setup instructions, intended for developers. Click here to view the server setup instructions.

Local Dev Environment Setup

1. Download php 7.4.27 x64 thread-safe zip - https://windows.php.net/download#php-7.4.
2. Extract the zip file and copy the whole folder to Program Files and add to your path - https://www.geeksforgeeks.org/how-to-install-php-in-windows-10/.
3. Create your php.ini file - In php folder, copy php.ini-development and paste (rename to php.ini).
4. Open php.ini, Ctrl-F and uncomment the following lines (remove ; so they look like the following):

```
; On windows:  extension_dir = "ext"
extension=curl
extension=fileinfo
extension=mbstring
extension=openssl
extension=pdo_mysql
```

5. Download and install Composer from https://getcomposer.org/download/ by using the Composer-Setup.exe with no proxy.
6. Download and install Node.js 16.3.2 LTS from https://nodejs.org/en/.

7. Open IDE of choice. We recommend PhpStorm.
8. Open IDE, then open the directory Laravel.
9. (PhpStorm only) Go to File -> Settings -> PHP and make sure you have set PHP Language Level 7.4 and CLI Interpreter 7.4.27.
10. Open a terminal window (PhpStorm has a terminal tab at the bottom) and make sure the current directory is Laravel.
11. Run *npm install* from the terminal.
12. Run *composer install* from the terminal.
13. Create a .env file. Copy .env.example (NOT .env.prod), then paste (rename to .env).
14. Download MySQL and MySQL Workbench to view tables and have a local database. https://dev.mysql.com/downloads/workbench/. Make sure your MySQL version is at least 8.0.28.
15. When the local database is set up, edit the DB_USERNAME and DB_PASSWORD in your .env file according to your local credentials.
16. Run the SQL statements in database/environmentSetup.sql to create the two necessary schemas.
17. Run *php artisan migrate* from the terminal with Laravel as the active directory to create all necessary tables.
18. Generate key, run *php artisan key:generate* from the terminal with Laravel as the active directory.
19. To run the Laravel project, run *php artisan serve* with Laravel as the active directory (keep this terminal up).
20. The project is available at http://127.0.0.1:8000/.

Updating the Theme

1. Make sure your bitbucket credentials are up-to-date in Laravel/auth.json.
2. Update the iastate-theme/laravel version in composer.json.
3. Run composer install from a terminal with Laravel as the active directory.
4. Run php artisan vendor:publish --force from a terminal.
5. Pick “Provider: IastateTheme\Laravel\ThemeServiceProvider”.
6. Git restore Laravel/config/theme.php. We don't want to change this file.
7. Commit the result.

Server Setup

The server is set up already, but if for some reason you want to run Transfer Pathways on a different server here are the instructions:

1. Download and install WAMP Server version >=3.2.6 from https://sourceforge.net/projects/wampserver/files/latest/download. Install into C:\wamp64.
2. Download upgrade to WAMP Server 3.2.7 from https://sourceforge.net/projects/wampserver/files/WampServer%203/WampServer%203.0.0/Updates/.
3. Download update to MySQL 8.0.28 from https://sourceforge.net/projects/wampserver/files/WampServer%203/WampServer%203.0.0/Addons/Mysql/.
4. Create folders Default, Deploy, and Backup in C:\wamp64\www.

5. Download the source code for this project and move the Laravel directory to C:\wamp64\www\Default.
6. Allow port 80 in the firewall: https://arcanecode.com/2018/01/02/opening-port-80-in-windows-firewall-to-support-calling-ssrs-from-another-computer/.
7. Make sure the WAMP Server is running (icon should be green). Go to Apache -> http-vhosts.conf and modify, so it looks like the following:

```
# Virtual Hosts
#
<VirtualHost *:80>
ServerName localhost
DocumentRoot "c:/wamp64/www/Default/Laravel/public"
ErrorLog "logs/transfer-pathways-error.log"
  <Directory "c:/wamp64/www/Default/Laravel/public/">
    Options +Indexes +Includes +FollowSymLinks +MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
    Require all granted
  </Directory>
</VirtualHost>
```

8. Change MySql version -> WAMP Icon -> MySQL -> Version and set to 8.0.28.
9. Download MySql Workbench from https://dev.mysql.com/downloads/workbench/.
10. Open MySql Workbench and connect to the already running WAMP Server to the MySql database at localhost.
11. In MySql Workbench, go to Server -> Users and Privileges, and create a user sdmay22_20.
12. Save the password you just created to a text file in C:\Database\dbpword.txt.
13. Run migrations, *C:\wamp64\bin\php\php7.4.26\php.exe artisan migrate --force*.
14. Install Gitlab Runner for auto deployment (Registration token can be obtained from https://git.ece.iastate.edu/sd/sdmay22-20/-/settings/ci_cd) by following the below steps:
15. Run PowerShell (as admin): https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/starting-windows-powershell?view=powershell-7#with-administrative-privileges-run-as-administrator.
16. Create a folder: C:\GitLab-Runner.
17. Enter the folder:

```
cd 'C:\GitLab-Runner'
```

18. Download the binary:

```
Invoke-WebRequest -Uri "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe" -OutFile "gitlab-runner.exe"
```

19. Register the Runner (steps below), then run:

```
./gitlab-runner.exe install -user=<usernameOfVM> -password=<passwordOfVM>
./gitlab-runner.exe start
./gitlab-runner.exe register --url https://git.ece.iastate.edu/ --registration-token <token>
```

The type of runner should be `shell` in the register step.

When you are finished, you should have a config.toml file.

Edit and under [[runners]] set *executor = "shell"*.

Server is now set up. But you should also set up a task to remove old guest data from the database (also currently set up):

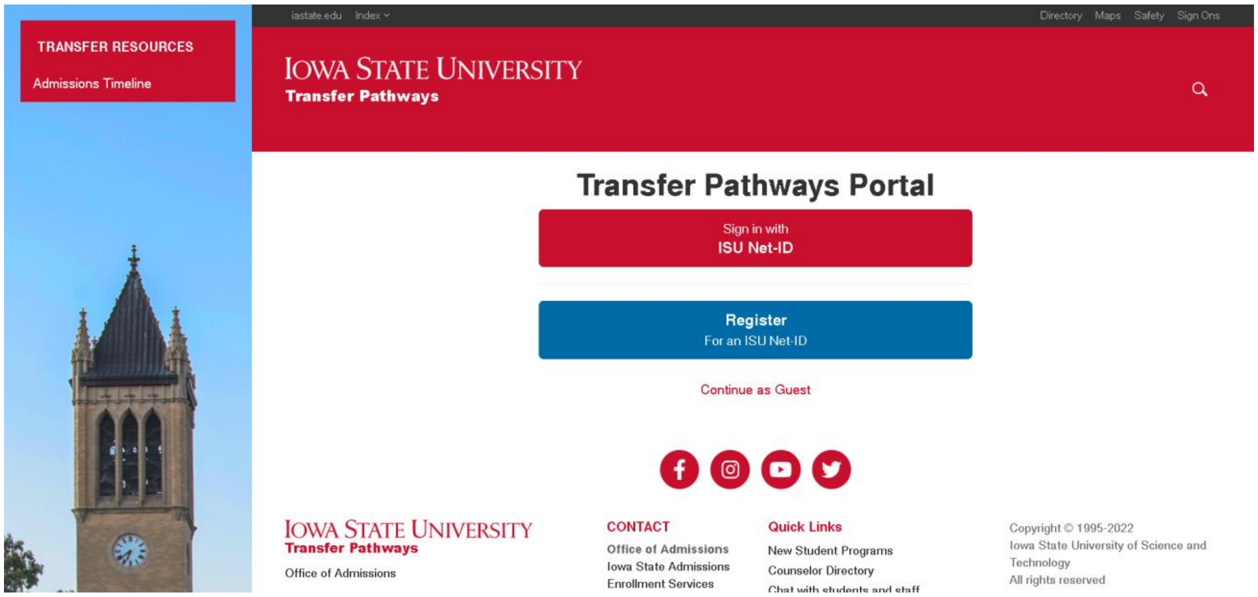
Guest data is stored in the entermajors and entercourses tables in the middlewaredb. Once a guest loses their session, they will not be able to access that saved data again. In order to keep the database clean, it is best to remove this old data daily by running the Laravel scheduler. This implementation of the Laravel scheduler has a command that can run up to every minute to remove old guest data. These steps walk you through how to set up a daily scheduled task to remove old guest data by running the DeleteOldMajorsAndCourses.bat file.

1. Go to Windows Task Scheduler (Press Win + R and type “taskschd.msc”)
2. Click "Create Basic Task" from the right-side menu
3. Enter the name and description as whatever you want and click next
4. Click "Daily" and click next
5. Set a time for the task to run daily and click next. (Ideally, this would be overnight when users aren't using the system)
6. Choose "Start a Program" and click next.
7. Under "Program/Script" browse for the .bat file called "DeleteOldMajorsAndCourses.bat" in the root of this project.
8. Click "Finish"
9. Check the "Open properties dialog" option and click "Finish"
10. Under the "General" tab choose "Run whether user is logged in or not"
11. Click "OK"

Demo Instructions

Transfer Pathways can currently be accessed at http://sdmay22-20.ece.iastate.edu/. **Note: you must be on the campus network or on the VPN in order to access the site.** Additionally, you can access the site locally if you have completed the local dev setup steps by going to http://127.0.0.1:8000/. However, this is only recommended for developers.

After reaching the site for the first time, you will see the Login page.


TRANSFER RESOURCES

Admissions Timeline

iastate.edu Index

Directory Maps Safety Sign Ons

IOWA STATE UNIVERSITY

Transfer Pathways

Transfer Pathways Portal

Sign in with ISU Net-ID

Register For an ISU Net-ID

Continue as Guest

IOWA STATE UNIVERSITY Transfer Pathways

Office of Admissions

CONTACT

Office of Admissions

Iowa State Admissions

Enrollment Services

Quick Links

New Student Programs

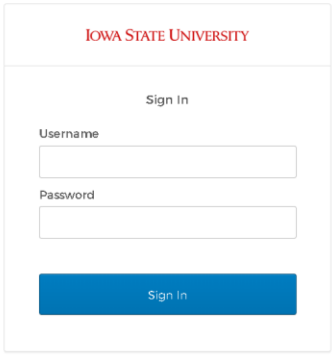
Counselor Directory

Chat with students and staff

Copyright © 1995-2022 Iowa State University of Science and Technology All rights reserved

Figure 30 - Login Page

At the Login page, you have three options, to sign in as a prospective student user, to sign in as an administrator user, or to continue as a guest. Additionally, you can click the blue button "Register for an ISU Net ID" to go to the Sign-Up page. It is of note that both sign in and registration functionalities are not currently connected to ISU's official Okta service. What that means is that you will not be able to use your current Net-ID to sign in and signing up will not allow you to sign in with the account you signed up as. If you wish to be a guest prospective student user, click on "Continue as Guest" and skip to the prospective student section of the demo. Otherwise, please click "Sign in with ISU Net-ID." You will be redirected to a prompt for credentials.


IOWA STATE UNIVERSITY

Sign In

Username

Password

Sign In

Figure 31 - Mock Otka Sign In Page

Again, please note that this is not the Okta you are used to. Do not enter your Net-ID credentials as they will be sent in plain text. Instead, here are the possible login credentials:

Administrator, Username: sdemoss, Password: goCyclones

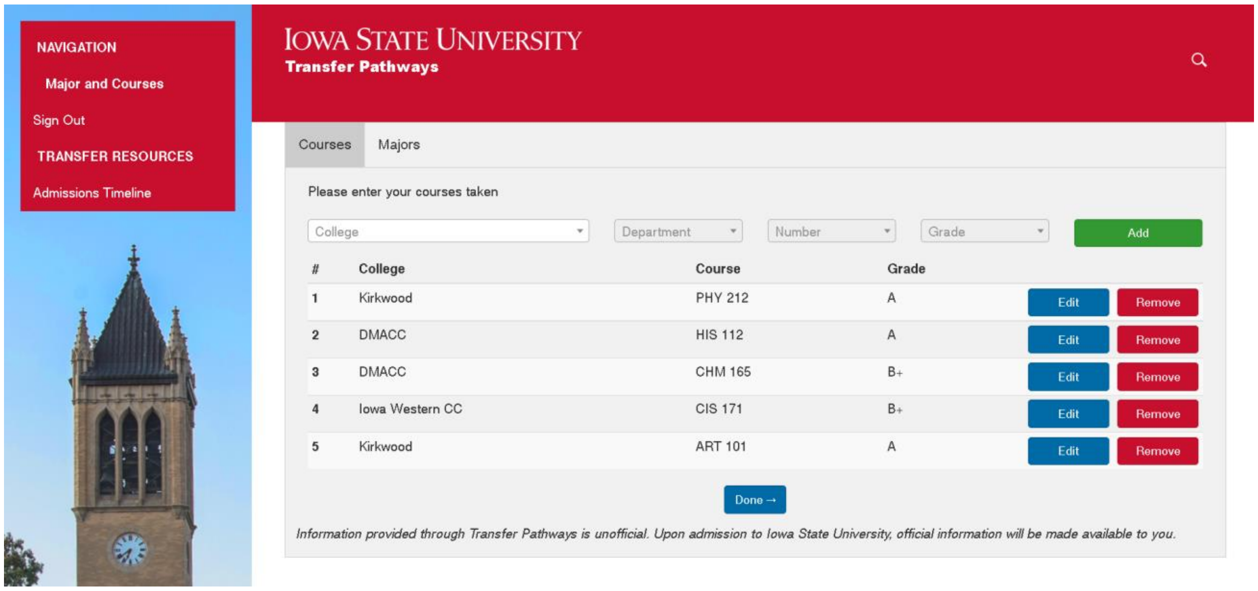
Prospective Student, Username: cy, Password: cyclonePower

Prospective Student, Username: mstewert, Password: ILoveCooking

Enter one of the combinations and press "Sign In." If you signed in as a student please continue to the below section, the prospective student demo. Otherwise, you are an administrator. You will be redirected to the home page for administrators, the Aggregate Data page. Please view the administrator demo in this case.

Prospective Student User Demo

After signing in as a prospective student user or continuing as a guest you will see the Majors and Courses page.



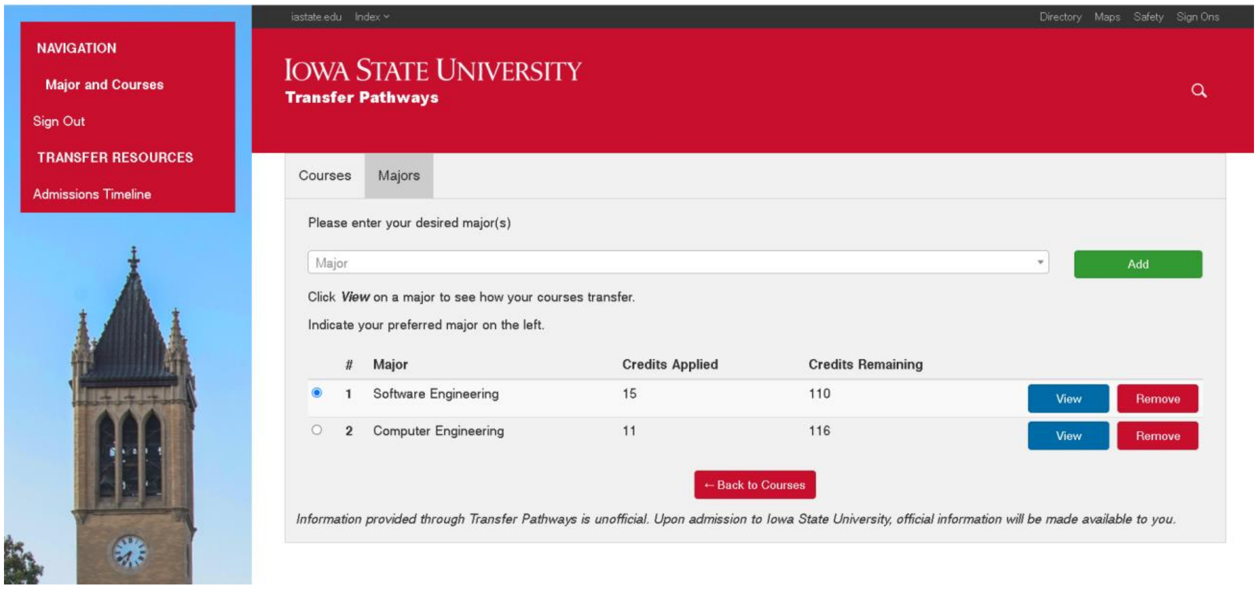
Please enter your courses taken

#	College	Course	Grade
1	Kirkwood	PHY 212	A
2	DMACC	HIS 112	A
3	DMACC	CHM 165	B+
4	Iowa Western CC	CIS 171	B+
5	Kirkwood	ART 101	A

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Figure 32 - Courses Tab of Majors and Courses Page

The "Courses" tab will be selected to start. On this tab, you can enter new courses. To enter a new course, click on the "College" dropdown and select a college. Note that all of the dropdown menus are searchable and you must make a selection before moving on to the next dropdown. Additionally, note that the courses and majors are mocked and not connected to live backend sources such as Workday or UAchieve. After selecting a college, move on to the department dropdown, and so on, until all dropdowns are filled out. Then, click "Add" to add the course. Additionally, you can edit the grade of an added course by clicking on the "Edit" button. To delete a course, click the "Remove" button next to the course you wish to delete. You will be prompted with a confirmation dialog. Finally, when you are ready to add your entered majors, please click "Done" or select the "Majors" tab.



Please enter your desired major(s)

Click *View* on a major to see how your courses transfer.

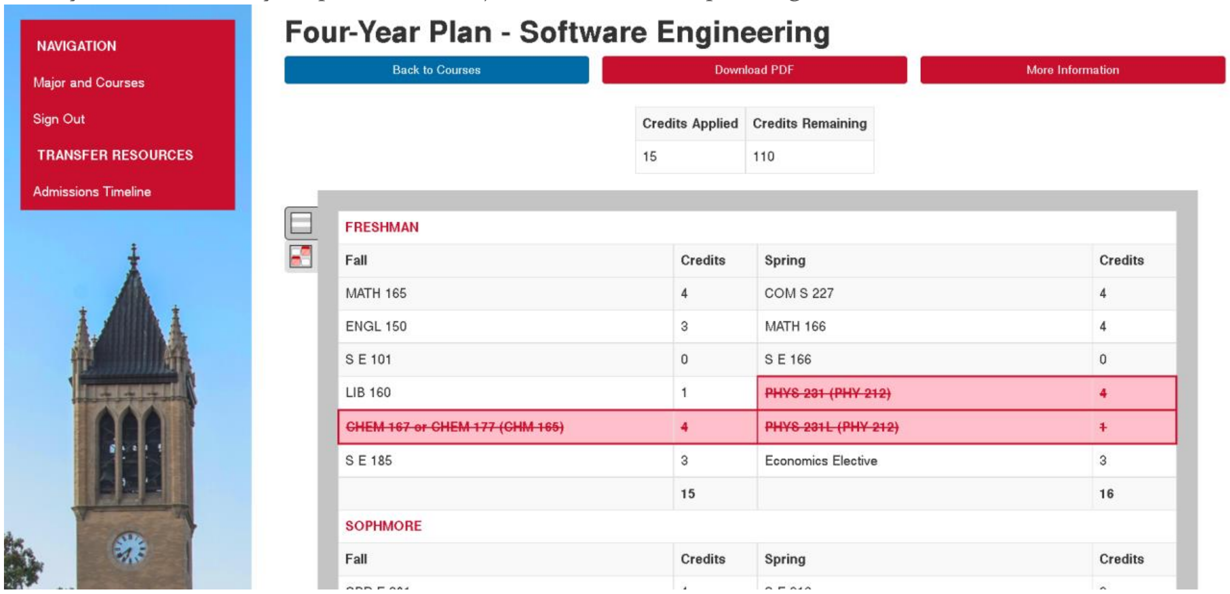
Indicate your preferred major on the left.

	#	Major	Credits Applied	Credits Remaining
●	1	Software Engineering	15	110
○	2	Computer Engineering	11	116

Information provided through Transfer Pathways is unofficial. Upon admission to Iowa State University, official information will be made available to you.

Figure 33 - Majors Tab of Majors and Courses Page

In the "Majors" tab, you are able to enter your desired majors to see how your credits transfer and view four-year plans. To add a major, click the "Major" dropdown and select a major (this dropdown is also searchable). Then, click "Add" to add the major. On the left side of the table, you can select the radio button to indicate your preferred major. This assists administrator users with determining which majors are the most popular. In the table, you are able to see the majors you have added, how many credits applied for each major, and how many credits are left for each major. Just like courses, majors can be deleted by clicking the corresponding "Remove" button. Finally, to view a four-year plan of the major, click the corresponding "View" button.



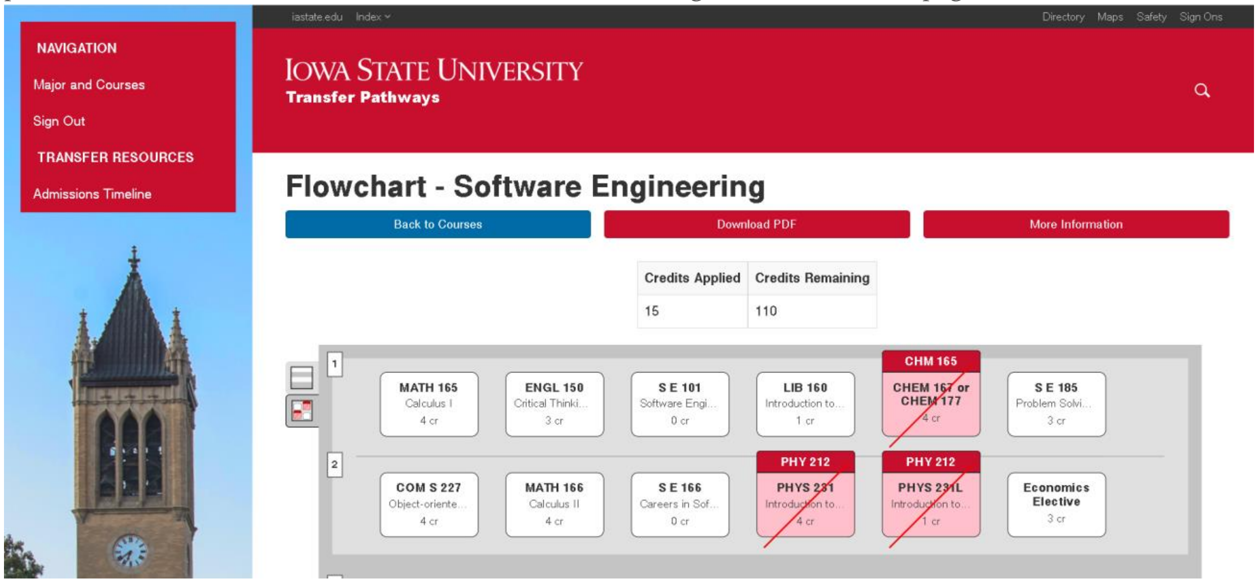
Four-Year Plan - Software Engineering

Credits Applied	Credits Remaining
15	110

FRESHMAN			
Fall	Credits	Spring	Credits
MATH 165	4	COM S 227	4
ENGL 150	3	MATH 166	4
S E 101	0	S E 166	0
LIB 160	1	~~PHYS 231 (PHY 212)~~	~~4~~
~~CHEM 167 or CHEM 177 (CHM 165)~~	~~4~~	~~PHYS 231L (PHY 212)~~	~~1~~
S E 185	3	Economics Elective	3
	15		16
SOPHMORE			
Fall	Credits	Spring	Credits

Figure 34 - Four-Year Plan Table Page

The Four-Year Plan Table page displays your customized four-year plan in table format. Courses you have successfully transferred are crossed off, while unfulfilled courses/requirements are not crossed off. In parenthesis next to the crossed off course/requirement is the name of the course you transferred from your previous institution. The “Download PDF” button allows you to download a PDF file of this table, while clicking “More Information” directs you to the Iowa State Catalog page for this major. At the bottom of the page, you can view the courses not applied to this four-year plan. Next, click on the small icon to the left of the table to go to the Flowchart page.



Flowchart - Software Engineering

Credits Applied	Credits Remaining
15	110

1

MATH 165 Calculus I 4 cr

ENGL 150 Critical Thinki... 3 cr

S E 101 Software Engi... 0 cr

LIB 160 Introduction to... 1 cr

CHM 165 ~~CHEM 167 or CHEM 177 4 cr~~

S E 185 Problem Solvi... 3 cr

2

COM S 227 Object-oriente... 4 cr

MATH 166 Calculus II 4 cr

S E 166 Careers in Sof... 0 cr

PHY 212 ~~PHYS 231 Introduction to... 4 cr~~

PHY 212 ~~PHYS 231L Introduction to... 1 cr~~

Economics Elective 3 cr

Figure 35 - Four-Year Plan Flowchart Page

This is the Four-Year Plan Flowchart page, which displays the same four-year plan information but in a flowchart format (without course prerequisites). The courses you have successfully transferred are red and crossed off, with the name of the course you transferred at the top of the rectangle. To see the full course name of a course, hover over (or tap on mobile) the rectangle of the course name you wish to see. The “Download PDF” button allows you to download a PDF file of this flowchart, while clicking “More Information” directs you to the Iowa State Catalog page for this major. At the bottom of the page, you can view the courses not applied to this four-year plan. You are now at the end of the prospective student section of the demo. If you wish to continue to the administrator demo, please click “Sign Out” on the navigation and sign in as an administrator.

Administrator User Demo

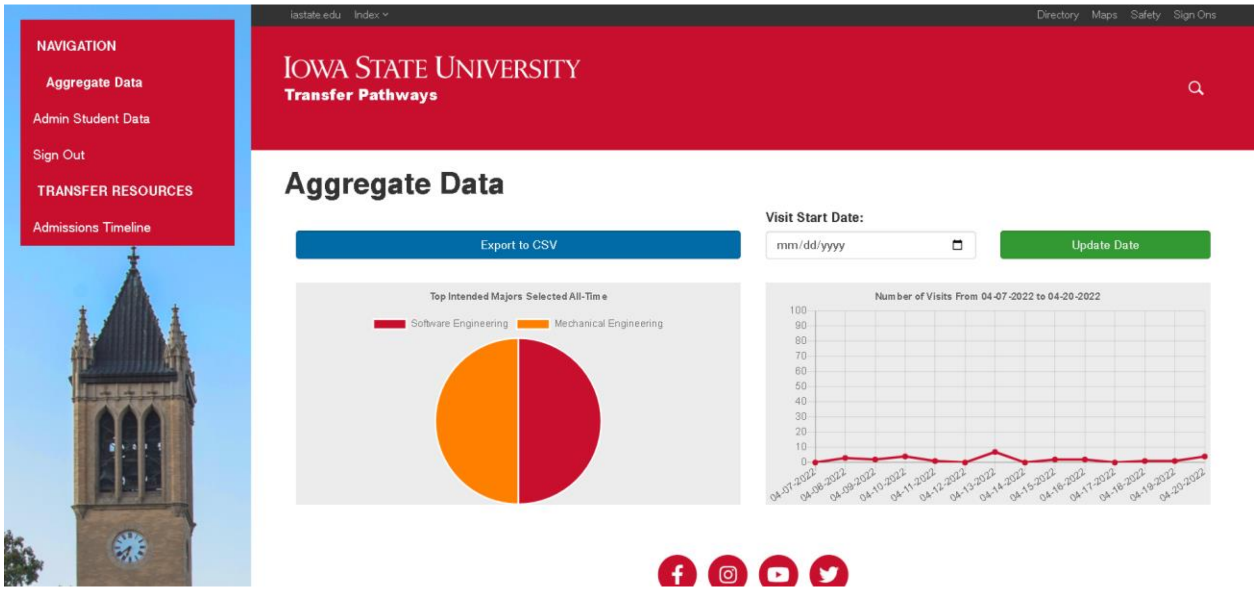
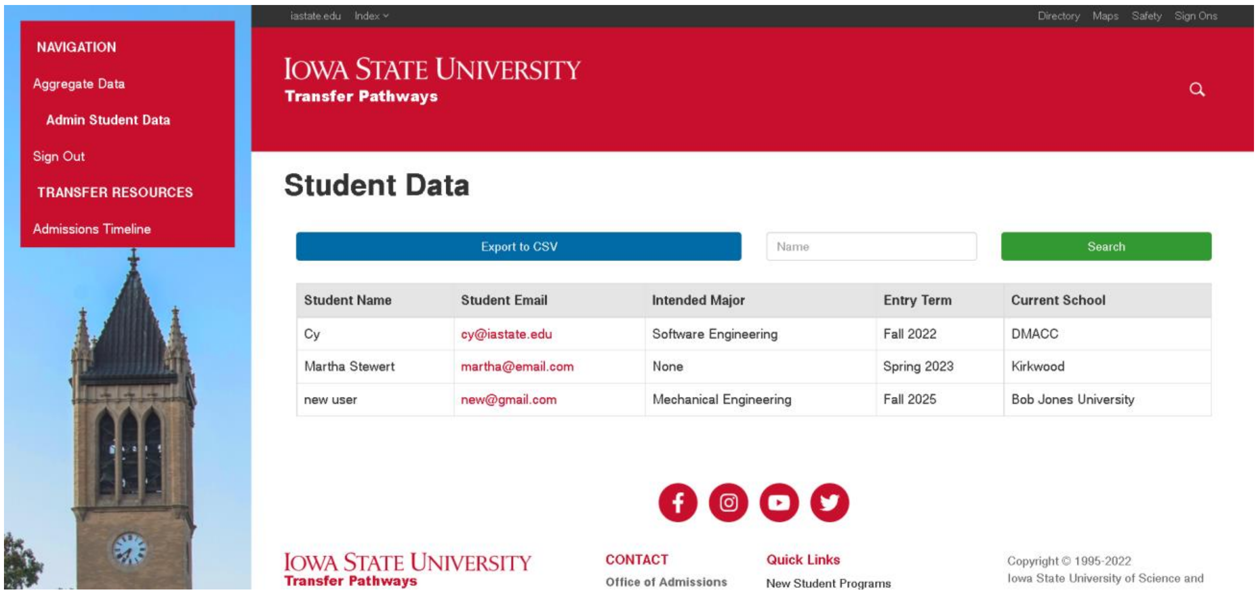


Figure 36 - Aggregate Data Page

On the Aggregate Data page, you can view aggregate data pertaining to all registered student users. The chart on the left displays the top seven intended majors selected by students. Click on the text of a major in the chart legend to cross it out. The chart on the right displays the number of visits in the past two weeks to the site by registered prospective students. Note that the visit start date can be modified in the top right corner by entering in a new date and clicking on "Update Date." Finally, both the top intended majors and visit data can be exported to a .csv file by clicking on the "Export to CSV" button. Note that when clicking on this button, you might get a warning about the site attempting to download multiple files; this is normal and is fine. Next, click on the "Admin Student Data" link in the navigation on the left side of the screen.



Student Name	Student Email	Intended Major	Entry Term	Current School
Cy	cy@iastate.edu	Software Engineering	Fall 2022	DMACC
Martha Stewert	martha@email.com	None	Spring 2023	Kirkwood
new user	new@gmail.com	Mechanical Engineering	Fall 2025	Bob Jones University

Figure 37 - Student Data Page

This is the Student Data page, which displays a table of individual student data. Much like the Aggregate Data page, the Student Data page is able to export the results as a .csv file. The textbox in the top right corner allows the user to search for a student by name. Finally, each student email is a hyperlink; clicking on that email will open up your default email client with a new message to that email address. This concludes the demo.

Testing Instructions

The following are instructions on how to run tests locally. To run tests using the CI/CD pipeline, please push to a branch in GitLab and create a merge request with the branch you just created. This will trigger the first two testing stages of the pipeline. Additionally, this pipeline will be rerun every time a push is made to the branch.

Note: Ensure dependencies are installed/dev environment set up, see Local Dev Environment Setup.

Unit Tests (phpunit)

Documentation: https://laravel.com/docs/7.x/testing.

To run unit tests run *php artisan test --testsuite=Unit* in a terminal with Laravel as the active directory.

To generate a unit test, run *php artisan make:test <testName> --unit* in a terminal with Laravel as the active directory.

JavaScript Unit Tests (Mocha)

Documentation: https://mochajs.org/.

Make sure you have run *npm install* recently in a terminal with Laravel as the active directory.

To run unit tests run *mocha --recursive 'tests/JS/*.js'* in a terminal with Laravel as the active directory.

Feature Tests

Documentation: https://laravel.com/docs/7.x/testing and https://laravel.com/docs/7.x/http-tests.

To run all feature tests run *php artisan test --testsuite=Feature* in a terminal with Laravel as the active directory.

To generate a feature test, run *php artisan make:test <testName>* in a terminal with Laravel as the active directory.

Browser Tests (Laravel Dusk)

Documentation: https://laravel.com/docs/7.x/dusk.

Testing framework (for example page-specific methods) should go in Browser/Pages.

If you want to run tests in headless mode locally, modify DuskTestCase.php.

Before running tests for the first time, run *php artisan dusk:chrome-driver <Your Chrome Version #>* with Laravel as the active directory.

<Your Chrome Version #> must match the whole number (for example 98) of your current Chrome version.

To generate a browser test, run *php artisan dusk:make <testName>* in a terminal with Laravel as the active directory.

First, serve the application, run *php artisan serve* in a terminal with Laravel as the active directory.

To run all browser tests run *php artisan test --testsuite=Browser* in a terminal with Laravel as the active directory.

Alternatively, run *php artisan dusk* in a terminal with Laravel as the active directory.

To run a specific test run *php artisan dusk --filter=<YourTestMethod>* in a terminal with Laravel as the active directory.

To run all failed tests run *php artisan dusk:fails* in a terminal with Laravel as the active directory.

Security Scanner (SonarQube)

One-time local setup - Steps taken from https://docs.sonarqube.org/latest/setup/get-started-2-minutes/ and https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/.

1. Download SonarQube Community Edition (latest) from https://www.sonarqube.org/success-download-community-edition/.
2. Unzip the contents of the zip you downloaded to C:\sonarqube (for example, you should have a C:\sonarqube\bin).
3. Open Powershell and enter the command C:\sonarqube\bin\windows-x86-64\StartSonar.bat.

4. If you have issues, refer to https://docs.sonarqube.org/latest/setup/troubleshooting/. It might be an issue with your version of Java.
5. Once you see SonarQube is up (it can take a while) go to http://localhost:9000/.
6. The credentials for first-time access are login: admin and password: admin. It will make you reset the password.
7. You should see the screen titled "How do you want to create your project?" Select Manually.
8. On the screen "Create a project" enter "Transfer Pathways Tool" under "Project display name". It will fill in the key. Press "Set Up."
9. For "How do you want to analyze your repository?" press Locally.
10. On "Analyze your project" under "Provide a token" type "main" and create. The token is now created. Save this. Press "continue".
11. Under "Run analysis on your project" select "Other". For "What is your OS?" select your OS.
12. Go to https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.7.0.2747-windows.zip to download the scanner.
13. Unzip the contents into C:\Program Files\SonarScanner (you should have C:\Program Files\SonarScanner\conf for example).
14. Go to C:\Program Files\SonarScanner\conf, open sonar-scanner.properties, and uncomment *sonar.host.url=http://localhost:9000*.
15. Add "C:\Program Files\SonarScanner\bin" to your path.

You have set up SonarQube. The only step you may have to repeat is step 3 if SonarQube is not running.

To run analysis:

1. Open Powershell and set the working directory to the git root for your project, NOT <git root>/Laravel.
2. Type *sonar-scanner.bat -D"sonar.login=<Saved Token from Step 10>"* to run analysis.
3. When complete, go to http://localhost:9000/dashboard?id=Transfer-Pathways-Tool to see the completed analysis.

Click "Security Hotspots" to see security issues.

APPENDIX II - Initial Version of Design

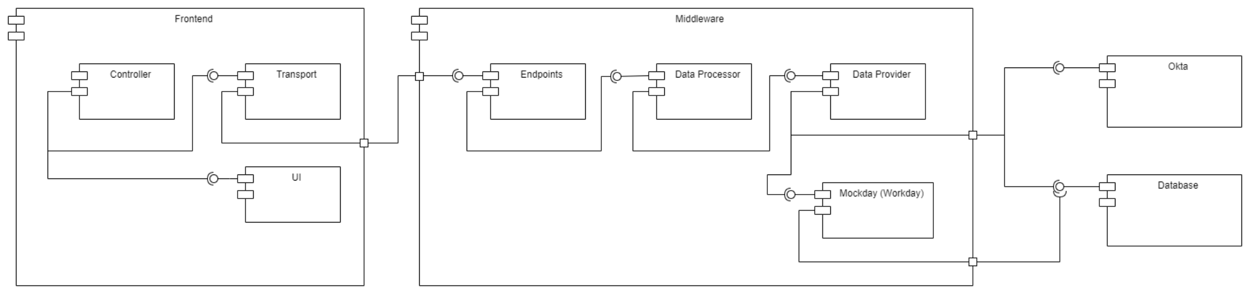


Figure 38 - Initial Component Diagram

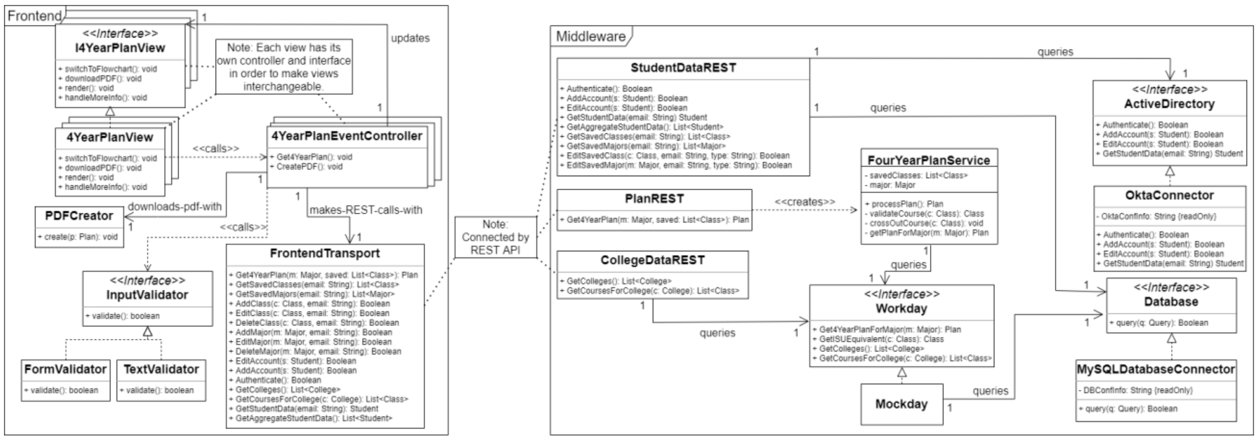

Frontend
<<Interface>>
I4YearPlanView
+ switchToFlowchart(): void
+ downloadPDF(): void
+ render(): void
+ handleMoreInfo(): void
Note: Each view has its own controller and interface in order to make views interchangeable.
updates
1
4YearPlanView
+ switchToFlowchart(): void
+ downloadPDF(): void
+ render(): void
+ handleMoreInfo(): void
<<calls>>
4YearPlanEventController
+ Get4YearPlan(): void
+ CreatePDF(): void
downloads-pdf-with
makes-REST-calls-with
PDFCreator
+ create(p: Plan): void
<<calls>>
<<Interface>>
InputValidator
+ validate(): boolean
FormValidator
+ validate(): boolean
TextValidator
+ validate(): boolean
FrontendTransport
+ Get4YearPlan(m: Major, saved: List<Class>): Plan
+ GetSavedClasses(email: String): List<Class>
+ GetSavedMajors(email: String): List<Major>
+ AddClass(c: Class, email: String): Boolean
+ EditClass(c: Class, email: String): Boolean
+ DeleteClass(c: Class, email: String): Boolean
+ AddMajor(m: Major, email: String): Boolean
+ EditMajor(m: Major, email: String): Boolean
+ DeleteMajor(m: Major, email: String): Boolean
+ EditAccount(s: Student): Boolean
+ AddAccount(s: Student): Boolean
+ Authenticate(): Boolean
+ GetColleges(): List<College>
+ GetCoursesForCollege(c: College): List<Class>
+ GetStudentData(email: String): Student
+ GetAggregateStudentData(): List<Student>
Note: Connected by REST API
Middleware
StudentDataREST
+ Authenticate(): Boolean
+ AddAccount(s: Student): Boolean
+ EditAccount(s: Student): Boolean
+ GetStudentData(email: String) Student
+ GetAggregateStudentData(): List<Student>
+ GetSavedClasses(email: String): List<Class>
+ GetSavedMajors(email: String): List<Major>
+ EditSavedClass(c: Class, email: String, type: String): Boolean
+ EditSavedMajor(m: Major, email: String, type: String): Boolean
queries
queries
PlanREST
+ Get4YearPlan(m: Major, saved: List<Class>): Plan
<<creates>>
CollegeDataREST
+ GetColleges(): List<College>
+ GetCoursesForCollege(c: College): List<Class>
queries
FourYearPlanService
- savedClasses: List<Class>
- major: Major
+ processPlan(): Plan
- validateCourse(c: Class): Class
- crossOutCourse(c: Class): void
- getPlanForMajor(m: Major): Plan
queries
<<Interface>>
Workday
+ Get4YearPlanForMajor(m: Major): Plan
+ GetISUEquivalent(c: Class): Class
+ GetColleges(): List<College>
+ GetCoursesForCollege(c: College): List<Class>
Mockday
queries
<<Interface>>
ActiveDirectory
+ Authenticate(): Boolean
+ AddAccount(s: Student): Boolean
+ EditAccount(s: Student): Boolean
+ GetStudentData(email: String) Student
OktaConnector
- OktaConfInfo: String {readOnly}
+ Authenticate(): Boolean
+ AddAccount(s: Student): Boolean
+ EditAccount(s: Student): Boolean
+ GetStudentData(email: String) Student
<<Interface>>
Database
+ query(q: Query): Boolean
MySQLDatabaseConnector
- DBConfInfo: String {readOnly}
+ query(q: Query): Boolean

Figure 39 - Initial Class Diagram

Figure 38 - Initial Component Diagram and Figure 39 - Initial Class Diagram above show our initial component and class diagrams respectively. The component diagram does not differ greatly from the final design, but the class diagram has changed to a greater degree. In retrospect, this is due mainly to the fact that a class diagram is highly dependent upon the technologies and frameworks it resides in. Our team, in general, had little experience with PHP and Laravel, so this aspect of the design was bound to change and adapt to better utilize the tools at our disposal. The basic principles and intents of our design, however, remained intact.

APPENDIX III - Other Considerations

Overall, we have learned a numerous amount throughout this project. First, only one member of the team had any experience with PHP development before so learning a PHP framework and the language in general was a challenge for many of us. We were able to learn a lot about how to code successfully within the PHP Laravel framework through either the documentation or code reviews by fellow team members who had experience with that aspect of PHP in a previous part of the project. We also learned a lot about security testing within software. One member of our team had taken a cybersecurity course, but the scale of security testing required for a project like this was much more than a basic understanding of cybersecurity. We learned how to use SonarQube to scan through our code and fix a few different vulnerabilities that it found as stated in section 4.7.2. This was important to make sure that we had secure code. Third, and perhaps most importantly, we learned how to communicate with our client and Iowa State University IT about the different requirements we had to fulfill and how to integrate our code with the Okta and future Workday services. This was extremely important for our project due to the fact that this project had strict requirements due to the nature of having to integrate with other software and replacing an already existing system that has many users that are familiar with it. We were able to hold a number of meetings with Iowa State IT and our client in order to deliver the best product possible. These meetings taught us a lot about the type of questions we should be asking which allowed us to grow as communicators. A couple examples of what we learned are that more pointed and direct questions tend to work better than open ended questions and to include examples. While these are the main things that we had to learn, some members of the team did have to learn basic things about software design principles, database management, and REST APIs in order to contribute effectively to the project. These were all key aspects of our project and each person individually caught up on what they didn't understand.

APPENDIX IV - Code

All code for this project can be found at the following repository: https://git.ece.iastate.edu/sd/sdmay22-20. It may also be retrieved from our client, Susie DeMoss, or Iowa State University Admissions IT.