DigiClips

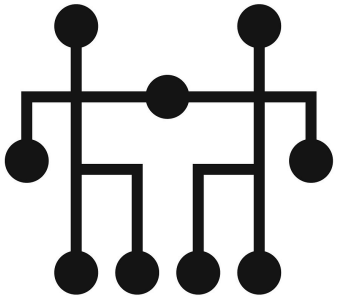
Modernization of the Administrative Website

sddec24-Team03

Jan 16, 2024 - Dec 20, 2024

https://sddec24-03.sd.ece.iastate.edu/

Faculty Advisor: Dr. Ashfaq Khokhar



Team Members/Role:

Derek Brandt - Project Manager / Architecture Lead

Israel Sanchez Tellez - Client Interaction Lead

Derek Brandt - Architecture Lead

Tyler Orman - Security Lead

Aryan Rao - Programming Lead

Danh Hoang - Programmer

Executive Summary

Project Overview

The Modernization of the Administrative Website aims to resolve the following identified issues with the previous Administrative portal:

- Outdated libraries.
- Messy and Unorganized User Experience.
- Tech Debt from comments and unimplemented features.

DigiClips mission is to develop a media recording system and sell it as a whole. Therefore, software quality needs to be high, and the cost of long-term maintenance should be low.

Summary of Requirements

The Development Requirements produce a higher quality product in the long term and are as follows:

- Services store and manage data, and UI manages user inputs.
- Features UI will live in a “box,” isolating its functionality.
- Master will never have commented-out code or unimplemented features.

The Functional and Aesthetic Requirements

- Display error messages, error statistics, and uptime statistics.
- Login credentials will be required for website access.
- Modify search engine account status and information.
- All graphs must include clear labels and titles.
- UI is a logical and intuitive structure.

UI Design and Software Architecture

The website uses **Angular** for its frontend and **Express** for its backend. Data is taken from a **shared MySQL database** that all of DigiClips uses. The user needs are broken down into **features which are implemented as ‘boxes,‘** and **boxes are combined to make dashboards**. Dashboards are all the functionality one user group will need.

Outcome

We completed a website that encompasses **all previous functionality**, makes use of **updated UI and libraries**, and has no **unused code or comments** (tech debt). The clients, the temporary users, had great success using and viewing the website.

Table of Contents

List of Figures and Tables

Figures

Tables

1. Project Overview

1.1. Who is DigiClips

DigiClips is a company set on creating a digital media system that can record data from outlets such as T.V. stations, radio channels, and more. The goal is to eventually develop a complete system that can be sold to a larger company.

1.2. Problem Statement

DigiClips relies on two critical subsystems: a customer-facing search engine and a data recording/processing server. Efficient access to key data—such as error logs, uptime metrics, server statuses, and machine configurations—is essential for maintaining operational reliability and addressing issues promptly. While an administrative system exists to facilitate access to this data, it has become outdated, inefficient, and overly complex. These flaws have introduced significant challenges for developers, increased maintenance costs, and reduced the product's overall value.

To address these challenges, our team will comprehensively modernize the administrative system. This effort will involve:

1. Library and Framework Updates: Transitioning to updated libraries and versioning practices.
2. System Optimization: Reducing complexity by removing redundant code and implementing clear object ownership.
3. User Experience Enhancements: Redesigning the interface to a modern, streamlined aesthetic while consolidating all core features into intuitive dashboards tailored to user roles.
4. Future-Proofing: Establishing clear development guidelines to simplify ongoing maintenance and ensure scalability.

We aim to enhance system efficiency, improve maintainability, and deliver a more robust and user-friendly administrative solution by implementing these measures.

1.3. Intended Users

There are four primary users we will target. The Administrator Website is an internal website that employees will access. Therefore, we will prioritize development toward Managers / Owners, Customer Support Representatives, and Backend Recording Engineers. These will be targeted through feature conversion/development and specialized dashboards (More on this in section 4. Project Design). Our project also involves a lot of

maintenance, so the final users we will target are the **Software Developers** working on the project.

1.3.1. Software Development Engineers

Profile:

- Age: 20–40 years old
- Expertise: Advanced knowledge in software development, coding, databases, and frameworks.

Needs:

- Documentation for compiling and executing the administrative website.
- Guidelines for future development practices.
- Code documentation outlining design patterns to follow.
- A roadmap of pending features and next steps.

Benefits:

Software development engineers do not directly use the website but develop it. Their primary benefit lies in adhering to well-documented practices that ensure the codebase is readable, maintainable, and easy to compile, ultimately reducing development time.

1.3.2. Managers / Owners

Profile:

- Age: 30–50 years old
- Expertise: Primarily business-focused with limited software engineering knowledge.

Needs:

- System monitoring for server and computer statuses.
- Access to user traffic statistics and machine uptime metrics.
- Tools to manage and control user permissions.

Benefits:

The administrative website provides managers with actionable insights into system performance and user interactions, enabling resource allocation and operational efficiency. Additionally, it helps prevent information misuse through robust user permission controls.

1.3.3. Customer Support Representative

Profile:

- Age: 30–50 years old
- Expertise: Minimal software experience, focused on client interaction.

Needs:

- Access to server statuses for accurate client updates.
- Tools to review customer account information.
- Logs of system errors and downtimes.

Benefits:

The administrative website streamlines access to customer and system information, significantly improving support efficiency compared to the previous cumbersome database access system. It enables immediate access to client account information, which is vital in customer support.

1.3.4. Media Recording Engineer

Profile:

- Age: 30–50 years old
- Expertise: Advanced knowledge of software systems, specializing in video, audio, and image processing backend systems.

Needs:

- Alerts and status updates on computers and subsystems in error states.
- Detailed error diagnostics for troubleshooting.
- Remote configuration management for system optimization.

Benefits:

The administrative website minimizes the effort required to identify and resolve system errors by consolidating error data and providing centralized access, reducing the overhead of manual database queries and system checks.

2. Requirements, Constraints, and Standards

2.1. Requirements & Constraints

The project requirements are organized into four categories: **Development Requirements**, **Functional Requirements**, **Aesthetic/UI Requirements**, and **Constraints**. Each category addresses specific aspects of the administrator website based on client specifications, standard practices, and project limitations.

2.1.1. Development Requirements

The development requirements are the responsibilities developers have to keep the project maintainable:

- Any HTTP Rest requests in the Angular project should be contained in an Angular Service.
- All future features will be developed as a 'box' unless a documented reason is given for larger features.
- Master will never have commented out code.
- Modules should either be tested or have a test plan prepped.

2.1.2. Functional Requirements

The functional requirements define the core capabilities of the administrator website:

- The website must display all error messages generated by backend machines.
- The website must display uptime statistics for backend machines.
- The website must allow modification of configuration files controlling backend machines.
- User login credentials will be required for website access.
- The website must provide functionality to modify user login information from the search engine.
- Super-administrators must have the ability to manage administrator accounts.

2.1.3. Aesthetic/UI Requirements

Aesthetic and UI requirements focus on ensuring the website is user-friendly, visually appealing, and aligned with standard UI design principles:

- All graphs must include clear labels and titles.
- User interface pages must follow a logical and intuitive structure.
- The website's visual design must be appealing and cohesive.

2.1.4. Constraints

Constraints define the technical and environmental limits within which the website must operate:

- 100% of current working features are converted into ‘boxes.’
- Must be able to handle ~2000 error messages at one time.

2.2. Engineering Standards

In order to ensure high-quality development and adherence to industry best practices, the project incorporates the following engineering standards, organized by their purpose and relevance to the administrative website.

2.2.1. Coding Standards

- Angular Coding Style Guidelines
    - Ensures uniformity in Angular code, improving readability and maintainability across the team.
- Google TypeScript Style Guidelines
    - Standardizes TypeScript code, enhancing collaboration and future development by ensuring consistency.

2.2.2. Discussed Standards with the Client

- HTTP/HTTPS Protocols
    - Purpose: Secure and reliable communication between the frontend and backend.
    - Application: All data transmission between the Angular frontend and backend server will use these protocols.
- Security Standards
    - Input sanitization and SQL stored procedures are used to protect database interactions.
    - Adherence to best practices prevents vulnerabilities and secures sensitive information.
- Performance Optimization
    - Techniques such as lazy loading, bundle size minimization, and optimized rendering improve user experience and system efficiency.
- Testing Standards
    - Implementation of unit and end-to-end tests for Angular applications ensures robustness and reliability.

2.2.3. Adopted Industry Standards

Standard	Purpose	Relevance
ISO/IEC 25010:2011	Framework for assessing software quality (e.g., functionality, reliability, maintainability).	Ensures the website is reliable, maintainable, and functionally suitable for DigiClips' needs.
ISO/IEC 27001:2013	Information security management standard.	Protects sensitive data through strong security policies and access controls.
IEEE 829-2008	Guidelines for software testing documentation.	Establishes structured testing plans, benefiting error monitoring and system reliability.
ISO/IEC 20000-1:2018	Service management system standard.	Provides structured service management for continuous system operation and incident response.
IEEE 14764-2006	Guidance on software maintenance processes.	Ensures a structured approach to updates, error corrections, and system enhancements.
ISO 9241-210:2010	Human-centered design standard for usability and accessibility.	Improves user experience by creating an intuitive interface for all user groups.

3. Project Plan

3.1. Project Mangement Style

Following industry practices, the project was broken into tasks using a work breakdown structure. Each work package then becomes a task that will get assigned to a member of the team. These tasks are completed through Agile planning. The sprints will last **two weeks**. We will operate under the idea that those two weeks will have the following:

- **8 hours of work for semester 1.**
- **15 hours of work for semester 2.**

The difference between semester one and semester two is the amount of classwork expected to be done outside of class. The designing and prep work is expected to take less time than implementation.

We will aim for **three points per sprint**. Agile was chosen for its adaptability. As will be briefed later, our main areas of risk lie in programming. Being adaptive to bugs and any other issues will be vital to the success of the project.

3.2. Progress Tracking

In order to track any work being done, the following sources should be updated bi-weekly:

- **Gantt Chart Progress Tracker** tasks are updated with the estimated percentage complete or EPC.
- **Clients Github** will be updated will commits and merge requests to track code progression. **Important:** GitHub progress is tracked by the client, so it is vitial that code updates be present. Ideally, more often than bi-weekly, but this is optional.

3.3. Task Decomposition and the Work Breakdown Structure

To represent the decomposed tasks, our team has elected to use a work breakdown structure. A full WBS combined Gantt chart is provided in *Appendix C*. We have provided a compressed WBS and Gantt chart for subgroups for convenience.

In order to read the WBS and Gantt chart, it is important to know some terminology. The **duration** represents the **estimated story points**, which are updated **on a bi-weekly basis** during sprint reviews. **EPC of Task Completed** represents the estimated percentage complete; however, all tasks that have an **EPC of 100% have been completed**.

3.3.1. Project Initiation and Analysis

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	EPC OF TASK COMPLETE	Initiation and Research							
					February				March			
					1	2	3	4	1	2	3	4
1	Project Initiation											
1.1	Contact Clients and Grab Project Information	Team	4	100%								
1.2	Develop Problem Statement	Derek B	2	100%								
1.2.1	Confirm Problem Statement with Clients	Israel S	2	100%								
1.3	Identify User Groups and Needs	Aryan R	2	100%								
1.4	Gather Requirements	Derek B	1	100%								
1.5	Gather industry examples and solutions	Tyler O	2	100%								
2	Previous Project Analysis											
2.1	Identify Technologies Used	Tyler O	2	100%								
2.2	Map Current Architecture	Derek B	2	100%								
2.3	Identify Current Features	Israel;Aryan	4	100%								

3.3.2. Redesign and Prototyping

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	EPC OF TASK COMPLETE	April			
					1	2	3	4
3	Redesign and Prototyping							
3.1	Redesign Feature into Boxes							
3.1.1	Machine Status Feature	Derek B	2	100%				
3.1.2	Error Table Feature	Tyler O	2	100%				
3.1.3	Verify Account Feature	Israel S	1	100%				
3.1.4	Block Account Feature	Tyler O	1	100%				
3.1.5	Error Graphs Feature	Israel S	2	100%				
3.1.6	Upload Config Feature	Aryan R	2	100%				
3.2	Compile designs for Dashboards							
3.2.1	Backend Analysis Dashboard	Tyler O	2	100%				
3.2.2	Customer Support Dashboard	Israel S	2	100%				
3.3	Prototype Design Structure in Angular	Aryan R	4	100%				
3.4	Redesign Code Architecture	Derek B	4	100%				

3.3.3. Implementation

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	EPC OF TASK COMPLETE	Implementation and Development									
					Aug.		September				October			
					3	4	1	2	3	4	1	2	3	4
4	Implementation													
4.1	Refamiliarate with the code base	Team	3	100%										
4.1.1	Identify Potential Tech Debt	Team	3	90%										
4.2	Delete commented and unused files	Tyler O	2	100%										
4.3	Update Dependencies													
4.3.1	Migrate from Angular 16 to Angular 18	Derek B	2	100%										
4.3.2	Migrate from Express 4 to Express 5	Derek B	1	100%										
4.4	Feature Conversation													
4.4.1	Convert Machine Status Feature	Derek B	2	100%										
4.4.1.1	Add Machine Status Service	Derek B	2	100%										
4.4.2	Convert Error Table Feature	Tyler O	3	100%										
4.4.3	Convert Verify User Feature	Danh H	2	100%										
4.4.4	Convert Block User Feature	Danh H	2	100%										
4.4.5	Convert Error Graph Feature													
4.4.5.1	Create Base Graph Interface	Israel S	1	100%										
4.4.5.2	Convert FFMPEG Error Tables	Israel S	2	100%										
4.4.5.3	Convert Failed To Record Tables	Aryan R	2	100%										
4.4.5.4	Improve Login Page UI	Tyler O	3	100%										
4.4.5.5	Convert Config Upload Feature	Aryan R	3	100%										
4.4.5.5	Create Error Service	Derek B	3	100%										
4.4.6	Combine for Backend Analyst dashboard	Danh H	2	100%										
4.4.7	Combine for Customer Support dashboard	Israel S	2	100%										
4.5	Reversed for Bug Fixes	Team	???	100%										

3.3.4. Project Completion and Deliverables

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	EPC OF TASK COMPLETE	Doc.		Project Wrap Up					
					May		November				Dec.	
					1	2	1	2	3	4	1	2
5	Project Completion and Deliverables											
5.1	Complete Sem.1 Review Presentation	Team	8	100%								
5.2	Review Sem.1 Project Doc.	Team	8	100%								
5.5	Review Sem.1 Progress with Clients	Team	4	100%								
5.3	Complete Comprehensive Test Plans											
5.3.1	Write Feature Test Coverage Plan	Team	8	70%								
5.3.2	Upload and communicate Test plan	Derek	1	100%								
5.4	Handoff Meeting with Oregon Team	Derek	1	100%								
5.6	Review Final Project Doc.	Team	8	100%								
5.7	Complete Final Review Presentation	Team	8	100%								
5.8	Finalize Final Review Poster	Team	4	100%								
5.9	Final Project Review with Clients	Team	4	100%								

3.4. Milestones and Metrics

For each subgroup, we also have identified milestones and metrics that we will use to track our progress.

3.4.1. Project Initiation and Analysis

- Gather and confirm 100% of client requirements and project scope.
- Complete a detailed problem statement and confirm it with clients.
- Identify 100% of user groups, their needs, and requirements.
- Map and analyze 90% of the current project architecture and features.

3.4.2. Redesign and Prototyping

- Redesign 100% of features identified, grouped into dashboard-specific components.
- Develop a functional structure prototype using Angular.
- Finalize redesigned code architecture for implementation readiness.

3.4.3. Implementation

- Clean the codebase by resolving 100% of identified tech debt, unused files, and outdated dependencies.
    - Zero commented code in master.
    - All functions are referenced at least once.

- Complete feature conversion and integration for all modules:
    - Machine Status
    - Error Table
    - Account Verification/Blocking
    - Error Graphs
    - Config Upload
- Integrate all converted features into the Backend Analyst and Customer Support dashboards.

3.4.4. Project Completion and Deliverables

- Develop a comprehensive use-case-based test plan for all features.
- Deliver the software to the client for integration testing on their machine.
- Finalize all project documentation.

3.5. Risk and Mitigation

3.5.1. Identified Risks and Mitigation Strategies

Mitigation techniques have been provided for all risks with either a probability of greater than 50% or a severity of High. The following is a list of identified risks.

- Missing critical client needs or misinterpreting requirements.
    - Risk Probability: **20%**
    - Severity: **High**
    - Mitigation Technique: Weekly meetings with clients will allow for progressive collaboration, ensuring needs and requirements are clear and agreed upon.
- Redesigns might not represent full functionality, leading to rework and tech debt for future teams.
    - Risk Probability: **30%**
    - Severity: **Medium**
- Upgrading Angular and Express might introduce bugs or compatibility issues.
    - Risk Probability: **60%**
    - Severity: **Medium**
    - Mitigation Technique: Time will be allocated at the end of the semester for any bugs discovered.
- Some features might take longer to convert than expected.
    - Risk Probability: **50%**
    - Severity: **High**
    - Mitigation Technique: Time allocated for a feature conversion will also include time to learn. This ensures that teammates can get a clear understanding with technologies and code they are unfamiliar with.

- The client is unable to integrate redesigned features.
    - Risk Probability: **40%**
    - Severity: **Medium**

3.5.2. Risks Encountered

The team ran into a few of the identified risks. Primarily, we ran into risk ***Some features might take longer to convert than expected***. However, the team was equipped to handle it as our schedule allowed us to buffer time. The remaining risks didn't come to bear as the team successfully converted all features, and weekly communication with the client kept all aligned in the designs.

In addition, the team also had to handle a missed risk. ***Failure to identify and estimate total previous tech debt.*** Adapting to the missed risk, the team created separate stories during the sprints to tackle the tech debt specifically. Cleaning up the debt after migrating the feature meant the requirements would be before any additional work was done.

3.6. Personnel Effort

Expected hours can be calculated by taking **eight weeks times sprint length**. To keep expectations within reason, we allowed for a week's worth of work as leeway for fluctuations in outside time commitments. A primary exception to this is Danh Hoang as he joined about halfway through the second semester.

In addition, the beginning of our first semester caused us to fall behind. We were unable to get access and set up a meeting due to technical issues and scheduling. However, we were able to push our plan back, as is reflected in *3.3. Task Decomposition and the Work Breakdown Structure*.

Teammate	Expect Hours (Semester 1)	Actual Hours	Expect Hours (Semester 2)	Actual Hours
Derek Brandt	64	43	105	120
Danh Hoang	0	0	55	54
Tyler Orman	64	44	105	107
Aryan Rao	64	43	105	99
Israel Sanchez	64	42	105	100

Table 1. Team Contribution Breakdown

3.7. Resource Requirements

3.7.1. Client-owned GitHub Repository

The client has provided a private Github that they request all teams use to keep track of their work so visibility is clear to both other teams and themselves. As such, our team will use this Github for our own personal code reviews and tracking.

3.7.2. Computer Environment

Each team member is expected to have a computer environment that allows them to build and run Angular and Express applications. This is required for both development and integration testing.

3.7.3. Package Manager

A common practice when developing TypeScript and JavaScript applications is to use a package manager to download remote dependencies. As the project will require third party libraries, a package manager is required to build the project. A recommended package manager is Node Package Manager or NPM.

4. Project Design

4.1. Broader Content

This project aims to resolve issues with the current administrative website that DigiClips has been developing. Since DigiClips aims to sell the entire system as a whole, the project aims to reduce the risk involved in taking on the administrative website as a product. Thus, the team hopes to improve the **economic** impact the project will pose.

4.2. Ideation and Design Decisions

The following design decisions have been taken from the previous design and market research. After client meetings, the team believes that these design decisions are in alignment and aim toward a better product for the client.

4.2.1. The administrative website will be a web application with user-specific dashboard layouts.

Based on market research, the team has decided to continue the development of the current administrative website; however, instead of using multiple pages, dashboards will be created with multiple features per page. Other solutions would considered, I.e., industry solutions like Tableau and Excel and different variations of custom websites.

The following decision matrix weighs each solution considered in the following categories on a scale of [0,5]: **Maintainability, Usability, Information Density (Info. Den.), and Cost**. The one exception is that information density is measured inversely. Ideas with a ton of information in one place would make it hard to read.

Idea	Maintainability	Usability	Info. Den.	Cost	Total
Dashboard Website	2	4	5	5	16
Display All Website	3	5	1	5	14
Multi Website	1	1	5	4	11
Excel	3	3	2	3	11
Tableau	4	4	3	2	13

Table 2. Decision Matrix for Project Format.

While the dashboard website had low maintainability due to potential tech debit introduced by student developers and the nature of building a custom solution, it was

highly ranked in the other categories. A custom website allows targeted information and targeted features. A dashboard website was more efficient at displaying information and features for each user group mentioned in section *1.3 Intended Users*.

4.2.2. All features will be isolated into a 'box' or a 'card.' Dashboards will be a suite of boxes.

In order to create a development environment that allows for cross-team collaboration, there must be an agreed-upon development workflow. Isolating features into boxes that will live in dashboards will create a low likelihood of inter-team conflicts. In addition, it will encourage visual resemblance between all current and new features. (See section *4.4. UI Designs and Box Definition* for more information on boxes.)

4.2.3. HTML Requests will only be made from an Angular Service.

Quoting from Angular's documentation, "Angular services provide a way for you to separate Angular app data and functions that can be used by multiple components in your app." Based on the team's analysis, this practice shall be followed as multiple features will share data. Also, this practice allows for "timed refreshes," a feature directly requested by the client.

4.2.4. All SQL requests will be made through Stored Procedures.

This is a direct request from the client. The team also performed an analysis of this decision and decided to adhere to it for two main reasons. First, previous implementations make use of stored procedures allowing us to reuse more code. Second, using stored procedures ensures that no SQL injections can be made to get data from unauthorized tables.

4.2.5. Github Master Branch contains only working features and no unused code.

Previously, unused (or commented) code was left in as teams ended their semester partially through feature development. This has led to confusion on which features have been implemented and which might be outdated. Thus, it has been decided that no unused code will be in the master branch, and it will only contain those features that function. Master sanitization is a common industry practice.

4.3. Previous Designs

Digiclips has been working on this project for multiple years. The team analyzed the previous design. These designs are illustrated in *Figure 1* and *Figure 2.*

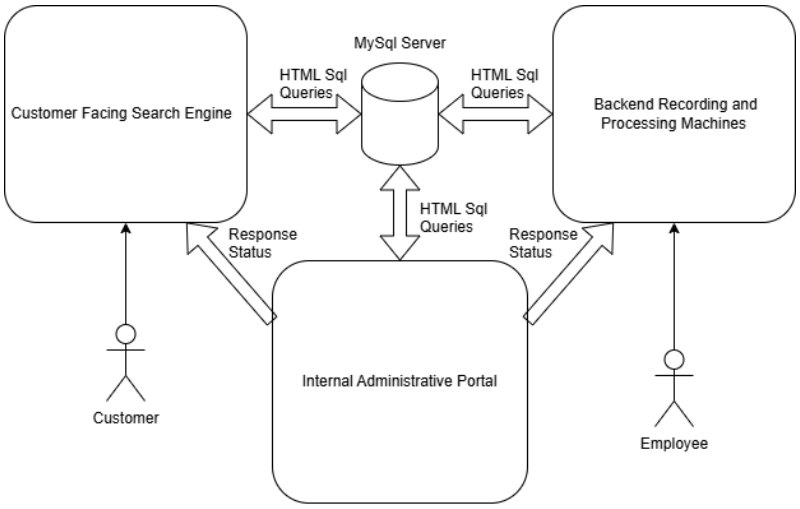


Figure 1. Complete Digiclips System Architecture

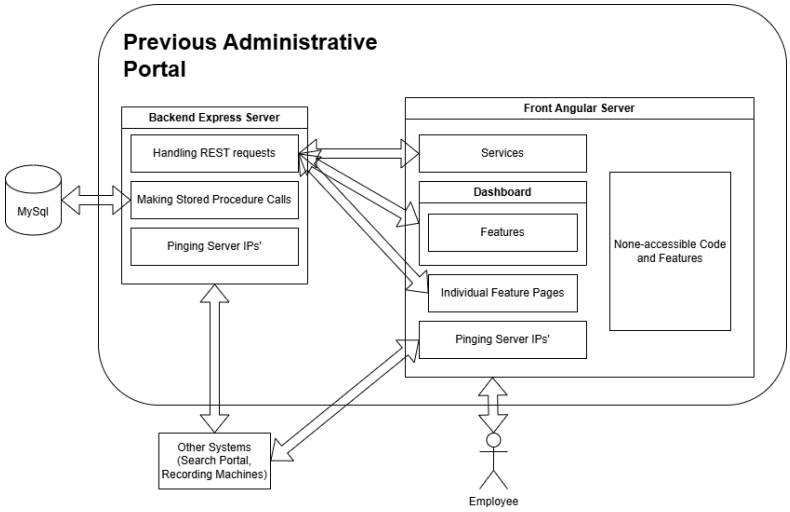


Figure 2. Previous Administrative Website System Architecture

This project targets moderizing the previous administrative website which is one of the three subsystems of DigiClips. The previous administrative website was a website built by multiple teams from multiple universities. As such, coding practices and features have changed or become stale. This is shown in the 'Not-Accessible Code and Features'. The design also contains redundant features and no clear rules or overall UI design.

4.4. System Architecture

The architecture resigns will only target the architecture of the administrative website. **The Express.js Backend** server will be responsible for the following:

- Inter-subsystem communication. I.e., SQL Queries and IP Pings.
- Handling REST Requests made from user frontends.
- Authentication requests.

The other subsystem, the **Angular.js Frontend** client handler, will be responsible for the following:

- Display graphical and tabular data.
- Uploading Configuration files.
- Handling user interaction.
- Managing User Accounts.

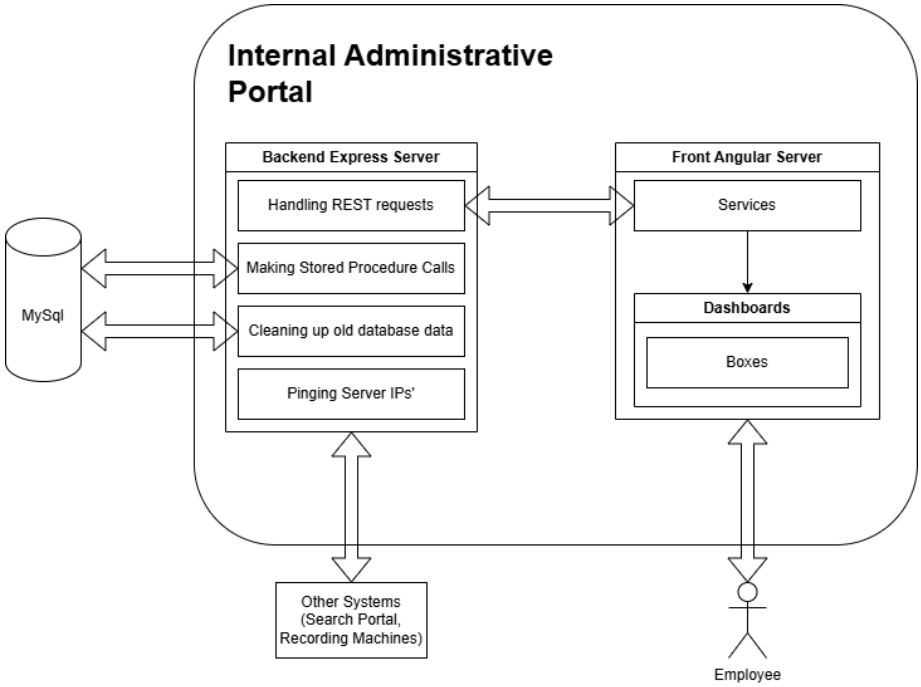


Figure 3. Modernized Administrative Website Architecture

Figure 3 depicts the visual representation of the architecture. The communication between the backend and frontend will be done through REST requests, and the SQL requests will be made through store procedures. The frontend is comprised of two primary parts: Services and Dashboards. Services handle storing data locally (data requested from the backend), and dashboards are the UI that display the data. For an example of a service, see *Appendix B*. Dashboards will then be comprised of boxes. Details regarding these will be included in the next section.

4.5. UI Designs and Box Definition

As mentioned in the previous section, the frontend is comprised of dashboards, which are comprised of boxes. Boxes isolate some single functionality, i.e., a graph box to display some error statistic or a site status (which is pictured in *Figure 4*).

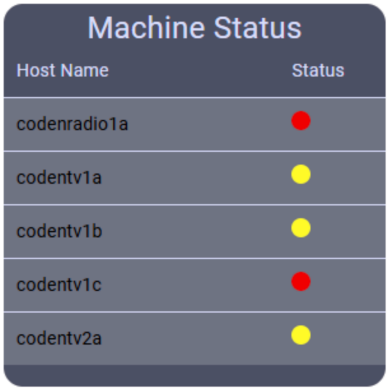


Figure 4. Machine Status Box.

The boxes are then grouped by functionality or user groups (listed in section *1.3 Intended Users*) *Figure 5* depicts an example of this dashboard. In this example, the backend analyst dashboard is depicted, which is comprised of the machine status, error table, upload config, and a couple of graphs. Each of these represents a single feature or statistic important to a backend analyst.

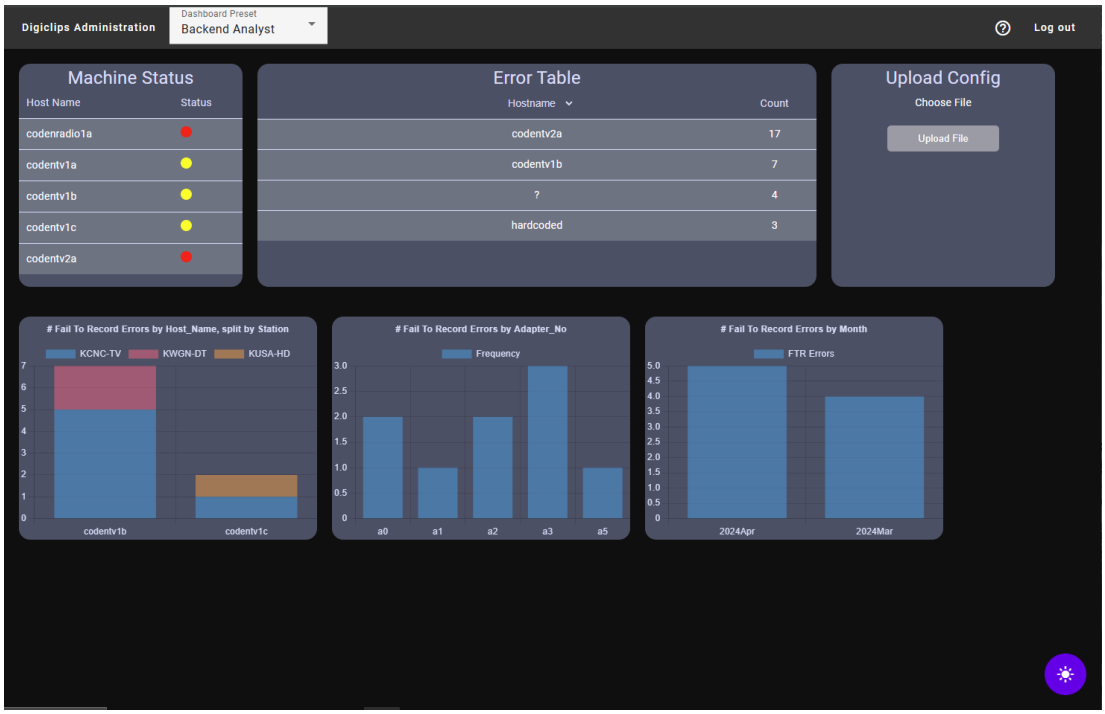


Figure 5. Backend Analyst Dashboard in Dark Mode.

4.6. Functionality

4.6.1. Developer Functionality

A developer can access the website differently than an end user. A developer will need to locally download the code base and **use a package manager** to download the website dependencies and build the project. A full detailed walkthrough is given in *Appendix A. Operational Manual*. There are two ways to run the project:

- **Backend Data Mocking** mocks data from the SQL database so the Express Backend can simulate its functionality.
- **Integration** requires a VPN to connect to allow the Express Backend to connect to the SQL database.

Before starting the website, the user needs to create a mock account. A script has been provided to do this, and an in-depth description is provided in *Append A. Operational Manual* and the ReadMe provided on Github.

Once the website is running locally, use should be simple. The user can select a dashboard through the dropdown menu shown in *Figure 6*. Once a dashboard is selected, the user is allowed to freely interact with each box on the dashboard.

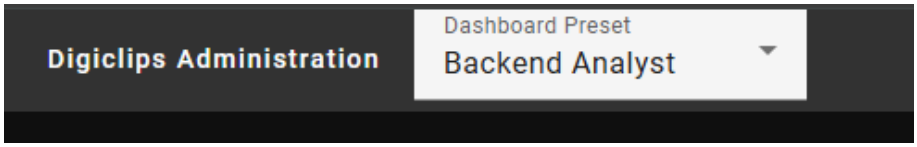


Figure 6 Dashboard Preset Dropdown.

4.6.2. Functionality for Future Deployment

This is an employee-only website. As such, all accounts should be managed by a manager and not provided through a 'Sign-Up.' These accounts will be stored in an SQL database.

The website will also run with a public IP address. This means that users can access the website without simulating it themselves and without remoting into a VPN. Due to the nature of this project, the public IP will be the responsibility of whoever buys the software.

4.7. Technology Considerations

4.7.1. Coding Technologies

The following technologies have been used in some form throughout the project.

- Angular - The client chose Angular, but after further research for its component-based architecture, two-way data binding, and scalability, it is ideal for building dynamic, single-page applications with reusable components.
- Express - Selected for its lightweight, flexible design, enabling quick setup of RESTful APIs, middleware integration, and efficient handling of HTTP requests.
- Jasmine/Karma - Angular's built-in testing functionality depends on Jasmine and Karma. They provide human-readable tests and cross-platform (Chrome, Firefox) test runners.
- Jest - Preferred for its comprehensive testing capabilities, including mocking, and snapshot testing, ensuring reliable and maintainable JavaScript applications.

4.7.2. Database

MySQL was the chosen database for two reasons: the client had one established so cross-group teams could share data, and converting this database had no upside. MySQL is still a great industry solution, easy to connect to without duplicating data (VPN in this case), and provides efficient results.

4.8. Areas of Challenge

The primary challenge that was faced was underestimating tech debt that occurs over time (I.e., bad code, comments, old dependencies) and the team's experience in web development. Only one team member was versed in Angular. Inexperience left a lot of development time researching Angular, bug fixes, and TypeScript.

However, due to proper planning and estimating of story time, the requirements were still met (I.e., all working features were converted and the master branch cleaned). Most of the user needs were met; however, roadblocks in interdependent team collaboration led to features being partially completed.

5. Testing

5.1. Unit and Interface Testing

5.1.1. Express Testing Via Jest

Express Backend testing was done with **Jest**, a testing framework. **A unit is defined as a single route**, i.e., https://[website ip]/myRoute. Each route will have a set of unit tests corresponding to it. An example of this can be seen in *Appendix B*. Since the Express server connects to the database and the OS, these must be mocked during testing.

5.1.2. Angular Testing

Angular recommends using Karma and Jasmine to run tests. Jasmine allows writing tests in a human-readable way while Karma runs the tests, mocking different browsers. A **unit is defined as a singular service or box.** An example of a box unit test is provided in *Appendix B*.

5.1.3. Interface Testing - Dashboards and Suites

Interface testing was done by **combining unit tests into suites**. One box will have multiple tests and combine to make it's test suite. When these get combined to make a dashboard, unit tests are written to verify that the dashboard contains all required boxes; however, testing the functionality of those boxes is left to their respective unit tests.

End-to-end tests are required to test the backend and frontend together. The team was unable to get Cypress functioning (Cypress is a testing framework commonly used for end-to-end testing.) We **ran manual tests** following user scenarios on mocked database data to compensate.

5.2. Integration Testing

Integration tests are very similar to interface testing with one key difference: **data is not mocked from the database**. Database integration has to be tested separately in case data from the database does not match the mocked data in the end-to-end testing. Just like interface testing, **integration testing is done manually**, following the same user scenarios done as interface testing.

5.3. System and Regression Testing

To verify the system behaves as expected, the combination of unit tests and (manual, but preferred automated) end-to-end testing. Unit tests are written based on common user scenarios for each box, and manual end-to-end (interface and integration) testing ensures that the system functions according to the requirements.

All unit tests are run (Frontend and Backend) before merging into the master branch. Manual testing is also done and reported during meetings, finalizing the merge requests. These tests ensure that the systems perform as expected and no feature's functionality has regressed.

5.4. Acceptance and User Testing

Conveniently, acceptance and user testing were done through the same process. The client, who is also a user, **accepts our designs and implementations after demo videos during the weekly meetings**. After accepting the new changes, the client would simulate and provide feedback on the functionality.

Overall, the client was **satisfied with the functional and aesthetic requirements**; however, they were concerned with the packages being outdated. This is to be expected as packages need to be constantly updated and can go out of date at any time.

5.5. Results

The team **excelled in Integration, Acceptance, and User testing**. The client gave **full approval of the functionality and designs during each demo and the final product review**. This means we met our Acceptance and User testing. Integration was our primary method of testing as it ensured real data would function properly. During the final product review, a full demo was given, highlighting integrated functionality.

Unit and Interface testing is where the team fell short, and a lack of experience was evident. While the team was successfully able to write **unit tests covering 90% of newly developed code**, **old code was often missed** as no previous testing had been done. However, this has been communicated to the next team and recommended testing steps are supplied.

6. Implementation

The team focused on refactoring some existing components by decoupling the dependencies of each UI component. We did this to make them into modular components. By injecting modularity into the code, the team was able to implement our design that allows the display of relevant components based on the user's role. Below is a breakdown of the components the team refactored:

- Machine Status - Display the status of different machines; Show real-time updates of performance and health.
- Error Table - Tracks and displays errors from the machines. It now provides clear and organized information for quick troubleshooting.
- Verify Account Page - Modernized layout for easier account details look-ups.
- Graphs - Decoupled graphs into individual components for reusability.

With the components now modular, the team shifted focus to modernizing the user interface. Following the design approved by the client, we implemented several changes to enhance the UI. The result is a sleek and visually appealing design. Below are the key changes implemented to the user interface.

- Box Base Class - Each of the components above inherits from this class. Therefore, they all follow the same design and can be easily modified if the client needs to make any future modifications.
- Dark Mode and Help Component - Dark mode is a standard component in modern app design for user preference and accessibility. The help component offers user support and troubleshooting resources.

Part of our design included changes that will improve code maintainability and faster development for future engineers. Below are some of the changes we did:

- Cleaned Code Base - As each component was being created, any unused or redundant code was removed.
- Upgraded to Angular 18 - The application can now be developed using the latest Angular framework with Long Term Support.
- Removed Package Vulnerabilities - Updated packages and resolved dependencies conflicts. The application now builds without any vulnerabilities.

7. Professional Responsibility

This discussion is with respect to the paper titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, International Journal of Engineering Education Vol. 28, No. 2, pp. 416–424, 2012

7.1. Areas of Responsibility

Area of Responsibility	Definition	NSPE Canon	IEEE Code of Ethics	NSPE vs. IEEE Differences
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	(6) To undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations (3) To be honest and realistic.	While the NSPE focuses primarily on only doing work within ones area of competence, the IEEE code allows for work outside of their competence as long as there are proper disclosures.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	(7) To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;	The IEEE code does not specifically cover the how one should act for clients, although it does mention to correct errors and accept criticism.

Table 3. Area of Responsibility Chart

Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	(3) To be honest and realistic in stating claims or estimates based on available data.	The two codes are similar on this value. They both state to be honest and realistic without deceit.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	(1) To accept responsibility in making decisions consistent with the safety, health, and welfare of the public. (9) To avoid injuring others, their property, reputation, or employment by false or malicious action.	While the codes are similar on this value, the NSPE holds safety paramount while the IEEE code is more focused on accepting the responsibility to make a safe product and avoid injuring others.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	(7) To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors.	While the IEEE code does not specifically mention how to act for clients, it does focus on accepting criticism and correcting errors.

Table 3 Continued

Sustainability	Protect environment and natural resources locally and globally.	Adhere to sustainable principles to protect the environment.	(1) To disclose promptly factors that might endanger the public or the environment.	The NSPE code focuses on adhering to sustainable practices while the IEEE code focuses on reporting environmental dangers.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	(5) To improve the understanding of technology; its appropriate application, and potential consequences.	While both codes talk about furthering the profession, the NSPE code focuses more on the personal conduct as opposed to improving understanding with the IEEE code.

Table 3 Continued

7.2. Project-Specific Professional Responsibility Areas

Work Competence - High: Work Competence plays a large role in our project as the team is developing a piece of software that will be in use within DigiClips. We are performing highly in this area because we are producing our project within our deadlines and to the highest quality with the knowledge gained from our coursework.

Financial Responsibility - High: The area of Financial Responsibility applies partially to the project, while the team is striving to create a high-quality product with a realizable value, the costs of the project are not applicable as we are using non-commercial software and internal tools. Our team has been able to design a high-quality product with zero financial cost to the client.

Communication Honesty - High: Communication Honesty is a high priority for our team. Working closely with the clients ensures that we are creating a product that will suit their needs and be useful to their company. Our team conducts weekly meetings with our clients in order to review designs and give general status updates on the project.

Health, Safety, Well-Being - NA: As our team's project is entirely software-based, Health, Safety, and Well-Being are not overly relevant. There are no significant Health, Safety, or Well-Being risks or concerns with our project.

Property Ownership - Medium: Property Ownership is relevant to our project and plays an important role in our team storing information regarding the clients of our application. Our team is working to ensure that our designs will properly handle any client information contained within the application.

Sustainability - NA: Due to our project being entirely software-based, the Sustainability value is not relevant to our project. Our project does not have a positive or negative impact on the environment or natural resources.

Social Responsibility - NA: Similarly, since our project is to be used as an internal tool, the Social Responsibility value is also not relevant to our project. Our project does not have a positive or negative impact on society or communities.

7.3. Most Applicable Professional Responsibility Area

Communication Honesty is the most applicable area to our project. Our team is focused on maintaining consistent communication with the clients to ensure a project is developed that suits their specifications. In addition to client communications, our team aims to develop and maintain proper code documentation throughout the implementation phase of the project. This will ensure that future developers are able to understand the code and work with it. Finally, easy to understand instructions will be created in order to explain the proper uses and features of the application.

8. Conclusion

8.1. Summary of Progress

The main focus of our project was to redesign the existing administrative portal and clean up its code base. The following are the primary results of our project are as follows:

- Cleaned Codebase - A significant amount of existing code was removed from the codebase. The removed code was often either large blocks of commented out code or functions that were no longer used by the portal. The remaining code in the Master branch is utilized and tested.
- Feature Conversion - Every feature that was present in the old administration portal has been converted to the box-based design and added to the new portal. These features include the machine status, error table, file upload, user management, and information graphs. Each feature has also been tested to confirm that any existing functionality was not lost in the transition.
- Integration and Unit Testing - Throughout development, we ran manual integration tests with the production database and created unit tests for each box to ensure that the components are functioning as expected.
- Acceptance Testing - We conducted acceptance testing with the clients throughout the semester by utilizing weekly meetings where we were able to gather feedback on the implementation of our designs. The end product was also thoroughly tested by the client to ensure usability.

8.2. Value Provided

- Cleaner UI - The end goal for our clients is to sell this product to a future organization. By cleaning up the UI with consistent and easy-to-use interfaces, we significantly increased the product's marketability.
- Reduced Tech Debt - We also drastically reduced the amount of tech debt within the product by upgrading the project from Angular 16 to Angular 18 to ensure that the portal utilizes currently supported libraries.

8.3. Next Steps

The next steps for the Administration Portal include a handoff to the teams that will continue development and a list of recommendations for features that will improve the functionality and marketability of the portal.

8.3.1. Team Handoff

The future development of the Administration Portal has been handed off to a team from Oregon State University. We have met with their team to explain our design implementation and provided a list of recommended features for the project.

8.3.2. Recommended Features

- Error Purging - Automatically purging errors from the database after a set time period would clean up the UI of the Error Table box by removing old errors that have already been investigated.

- Modular Dashboards - In the current implementation, the end users cannot change the boxes shown on their dashboards. Adding a feature that enables users to modify their dashboards would allow for more modularity and improve the end-user experience.

- Persistent Accounts - The authentication for the Administrative Portal is currently handled locally, and the user accounts do not persist across devices. Storing hashed credentials in the database for authentication would allow user accounts to persist across different devices and instances of the Portal.

- Website Configuration - There does not currently exist an efficient way to modify the theme or styling on the Administrative Portal. Utilizing a configuration file or similar tool would simplify the process of changing the theme and increase the product's marketability.

9. References

[1] IEEE Standards Association “IEEE SA - IEEE/ISO/IEC P23026: Systems and Software Engineering -- Engineering and Management of Websites for Systems, Software, and Services Information” [Online]. Available: https://standards.ieee.org/ieee/23026/10425/ [Accessed Apr. 16, 2024]

[2] Angular “Angular coding style guide” [Online]. Available: https://angular.io/guide/styleguide [Accessed Apr. 16, 2024]

[3] Google “Google TypeScript Style Guide,” [Online]. Available: https://google.github.io/styleguide/tsguide.html [Accessed Apr. 16, 2024]

Appendices

Appendix A - Operational Manual

Development Environment Set Up

1. **Cloning the Repository and Navigating to Project Root**
   - 1.1. Using Git Bash, clone the repository and navigate to the project root directory (./Admin).
   - 1.2. Verify the directory using (pwd).

git clone <repository-url>

cd ./Admin

pwd

2. **Install Backend Dependencies**
   - 2.1. Navigate to the backend directory (backend).
   - 2.2. Install the necessary dependencies using (npm install).
   - 2.3. Return to the project root.

cd backend && npm install && cd -

3. **Install Frontend (Angular) Dependencies**
   - 3.1. Navigate to the frontend directory (admin-ang).
   - 3.2. Install the necessary dependencies using (npm install).
   - 3.3. Return to the project root.

cd admin-ang && npm install && cd -

4. **Running the Environment**
   - 4.1. To prepare to run the environment, open up two terminals—one for the front and backend.
5. **Starting the Frontend**
   - 5.1. On the first terminal, navigate to the frontend directory (admin-ang).
   - 5.2. To run it on the localhost use (ng serve)
   - 5.3. This will start an instance of the frontend at http://localhost:4000

cd admin-ang

ng serve

6. Credential Creation

6.1. On the second terminal, navigate to the backend (backend).

6.2. Run the following command (node createAuthConfig.js).

6.3. Then, you will be prompted to enter an email and password. These credentials are saved in '.config.js' and will be the credentials to log into the site.

cd backend

node createAuthConfig.js

<enter email & password>

7. Starting the Backend

7.1. You can select one of the following to .js to run on the second terminal.

7.1.1. **'indexDev.js':** this file uses mock data.

7.1.2. **'index.js':** this file will attempt to use the real database.

7.2. The selected file will start the instance of the backend at http://localhost:8080

7.3. Run the following command to execute the file.

node src/index<Dev>.js

Appendix B - Code

Example of Angular Service (Machine Status Service)

```
@Injectable({
 providedIn: 'root'
})
export class MachineStatusService {
 private machineStatusSubject = new BehaviorSubject<MachineStatus[]>([]);
 public machineStatus$: Observable<MachineStatus[]> =
this.machineStatusSubject.asObservable();

 constructor(private httpService : HttpClient) {this.fetchMachineStatus()}

 fetchMachineStatus() {
   try {
     this.httpService.get<Array<Object>>(environment.serverUrl +
"/GetMachineCheckInStatus").toPromise().then(checkInStatus => {
       let convertedMachineStatus: MachineStatus[] = []
       checkInStatus.forEach((machine) => {
         convertedMachineStatus.push({
           host_name: machine['Host_name'] as string,
           status: machine['Status'] === "1" ? StatusType.Active :
StatusType.Pending
         });
       });

       for(let i = 0; i < convertedMachineStatus.length; i++) {

this.pingMachine(convertedMachineStatus[i].host_name).then(machineOnline => {
             convertedMachineStatus[i].status =
               StatusType.Pending === convertedMachineStatus[i].status &&
machineOnline ? StatusType.Reachable : StatusType.Error
         })
       }

       this.machineStatusSubject.next(convertedMachineStatus);
     })
   } catch (error) {
     console.error('Error fetching dashboard procedures:', error);
   }
 }
}
```

Example of Box HTML (Machine Status Box)

```
<div class="box" [ngStyle]="{
 'width': width,
 'height': height,
 'background-color': backgroundColor,
 'border-radius': borderRadius
}">
 <h3 class="title">Machine Status</h3>
 <div class="table-container">
 <table id="machineTable" class="status-table">
   <thead class="status-table-header">
     <tr>
       <th>
         Host Name
       </th>
       <th>
         Status
       </th>
     </tr>
   </thead>

   <tbody>
     <ng-container *ngFor="let machineStatus of statuses; let i = index">
       <tr (click)="setExpandedRow(i)" style="border-top: 1px solid #DBDFFE;">
         <td>
           {{machineStatus.host_name}}
         </td>
         <td>
           <div class="status-circle"
[ngClass]="getStatusClass(machineStatus.status)"></div>
         </td>
       </tr>

       <tr class="expanded-row">
         <td colspan="5" >
           <div class="expandable-content" [ngClass]="{'expanded': expandedRow
=== i}">
             <p style="padding: 10px">{{ getStatusMessage(machineStatus) }}</p>
           </div>
         </td>
       </tr>
     </ng-container>
   </tbody>
 </table>
 </div>
</div>
```

Example of Box TypeScript (Machine Status Box)

```
@Component({
 standalone: true,
 selector: 'app-machine-status-box',
 templateUrl: './machine-status-box.component.html',
 styleUrls: ['./machine-status-box.component.css'],
 imports: [ CommonModule ]
})
export class MachineStatusBoxComponent extends BoxBaseComponent implements
OnInit {
 constructor(private machineStatusService : MachineStatusService) { super() }
 statuses: MachineStatus[] = []
 expandedRow = null;
 ngOnInit(): void {
   this.machineStatusService.machineStatus$.subscribe(value => {
     this.statuses = value;
   });
   this.machineStatusService.fetchMachineStatus()
 }

 setExpandedRow(index: number): void {
   this.expandedRow = this.expandedRow === index ? null : index;
 }

 getStatusClass(status: StatusType): string {
   switch (status) {
     case StatusType.Active:
       return 'green';
     case StatusType.Reachable:
     case StatusType.Pending:
       return 'yellow'
     default:
       return 'red';
   }
 }
   getStatusMessage(status: MachineStatus): string
   {
     switch (status.status) {
       case StatusType.Active:
         return status.host_name + " has recently reported a check in!";
       case StatusType.Reachable:
       case StatusType.Pending:
         return status.host_name + " was able to be pinged; however, it has
not checked in!"
       default:
         return status.host_name + " has not checked in nor is it able to be
pinged. Resolve immediately!";
     }
   }
```

Example of Box Unit Test (Machine Status Box)

```
it('should call fetchMachineStatus on initialization', () => {
 spyOn(mockService, 'fetchMachineStatus');
 component.ngOnInit();
 expect(mockService.fetchMachineStatus).toHaveBeenCalled();
});

it('should toggle expandedRow when setExpandedRow is called', () => {
 component.setExpandedRow(1);
 expect(component.expandedRow).toBe(1);

 component.setExpandedRow(1); // Toggle off
 expect(component.expandedRow).toBeNull();
});

it('should return the correct class for each status', () => {
 expect(component.getStatusClass(StatusType.Active)).toBe('green');
 expect(component.getStatusClass(StatusType.Reachable)).toBe('yellow');
 expect(component.getStatusClass(StatusType.Pending)).toBe('yellow');
 expect(component.getStatusClass(StatusType.Error)).toBe('red');
});

it('should return the correct status message', () => {
 const activeStatus: MachineStatus = { host_name: 'Machine 1', status:
StatusType.Active };
 const reachableStatus: MachineStatus = { host_name: 'Machine 2', status:
StatusType.Reachable };
 const unreachableStatus: MachineStatus = { host_name: 'Machine 3', status:
StatusType.Error };

 expect(component.getStatusMessage(activeStatus)).toContain('has recently
reported a check in!');
 expect(component.getStatusMessage(reachableStatus)).toContain('was able to be
pinged');
 expect(component.getStatusMessage(unreachableStatus)).toContain('has not
checked in');
});

it('should render the table with statuses', () => {
 const rows = fixture.debugElement.queryAll(By.css('tbody
tr:not(.expanded-row)'));
 expect(rows.length).toBe(3); // One row per machine status

 const firstRowCells = rows[0].queryAll(By.css('td'));
 expect(firstRowCells[0].nativeElement.textContent.trim()).toBe('Machine 1');
});
```

Example of Router (Machine Check-In Route)

```
router.get("/GetMachineCheckInStatus", (req, res, next) => {
   var connection = db.getPool();
   connection.query(
       ("CALL dc." + "Fetch_TVMonitorStatus"),
       [req.body.Host_Name], //SELECT * FROM dc.Errors ORDER BY Date_Time DESC
LIMIT 30
       (error, results) => {
           if (error) {
               console.error(error);
               res.status(500).json({ status: 'error' });
           } else {
               res.status(200).json(results[0]);
           }
       }
   )
})
```

Example of Router Unit Test (Machine Check-In Route)

```
it('should return 200 and the results on success', async () => {
   const mockResults = [{ status: 'ok' }];
   __mockQuery.mockImplementation((query, params, callback) => {
       callback(null, [mockResults]); // Pass null for error, and mock results
for data
   });

   console.log("Waiting...")
   const response = await request(app)
       .get('/GetMachineCheckInStatus')
       .send({ Host_Name: 'test-host' });
   expect(response.status).toBe(200);
   expect(response.body).toEqual(mockResults);
   expect(__mockQuery).toHaveBeenCalledWith(
       'CALL dc.Fetch_TVMonitorStatus',
       ['test-host'],
       expect.any(Function)
   );
});
```

Appendix C - Gantt Chart

PROJECT TITLE	Modernization of Administrative Website	COMPANY NAME
PROJECT MANAGER	Derek Brandt	Last Updated

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	EPC OF TASK COMPLETE
				Initiation and Research: February 1 2 3 4; March 1 2 3 4
				Redesign: April 1 2 3 4; Doc.: May 1 2
				Implementation and Development: Aug. 3 4; September 1 2 3 4; October 1 2 3 4
				Project Wrap Up: November 1 2 3 4; Dec. 1 2
1	Project Initiation			
1.1	Contact Clients and Grab Project Information	Team	4	100%
1.2	Develop Problem Statement	Derek B	2	100%
1.2.1	Confirm Problem Statement with Clients	Israel S	2	100%
1.3	Identify User Groups and Needs	Aryan R	2	100%
1.4	Gather Requirements	Derek B	1	100%
1.5	Gather industry examples and solutions	Tyler O	2	100%
2	Previous Project Analysis			
2.1	Identify Technologies Used	Tyler O	2	100%
2.2	Map Current Architecture	Derek B	2	100%
2.3	Identify Current Features	Israel;Aryan	4	100%
3	Redesign and Prototying			
3.1	Redesign Feature into Boxes			
3.1.1	Machine Status Feature	Derek B	2	100%
3.1.2	Error Table Feature	Tyler O	2	100%
3.1.3	Verify Account Feature	Israel S	1	100%
3.1.4	Block Account Feature	Tyler O	1	100%
3.1.5	Error Graphs Feature	Israel S	2	100%
3.1.6	Upload Config Feature	Aryan R	2	100%
3.2	Compile designs for Dashboards			
3.2.1	Backend Analysis Dashboard	Tyler O	2	100%
3.2.2	Customer Support Dashboard	Israel S	2	100%
3.3	Prototype Desgin Structure in Angular	Aryan R	4	100%
3.4	Redesign Code Architecture	Derek B	4	100%
4	Implementation			
4.1	Refamiliarate with the code base	Team	3	100%
4.1.1	Identify Potential Tech Debt	Team	3	90%
4.2	Delete commented and unused files	Tyler O	2	100%
4.3	Update Dependencies			
4.3.1	Migrate from Angular 16 to Angular 18	Derek B	2	100%
4.3.2	Migrate from Express 4 to Express 5	Derek B	1	100%
4.4	Feature Conversion			
4.4.1	Convert Machine Status Feature	Derek B	2	100%
4.4.1.1	Add Machine Status Service	Derek B	2	100%
4.4.2	Convert Error Table Feature	Tyler O	3	100%
4.4.3	Convert Verify User Feature	Danh H	2	100%
4.4.4	Convert Block User Feature	Danh H	2	100%
4.4.5	Convert Error Graph Feature			
4.4.5.1	Create Base Graph Interface	Israel S	1	100%
4.4.5.2	Convert FFMPEG Error Tables	Israel S	2	100%
4.4.5.3	Convert Failed To Record Tables	Aryan R	2	100%
4.4.5.4	Improve Login Page UI	Tyler O	3	100%
4.4.5.5	Convert Config Upload Feature	Aryan R	3	100%
4.4.5.5	Create Error Service	Derek B	3	100%
4.4.6	Combine for Backend Analyst dashboard	Danh H	2	100%
4.4.7	Combine for Customer Support dashboard	Israel S	2	100%
4.5	Reversed for Bug Fixes	Team	???	100%
5	Project Completion and Deliverables			
5.1	Complete Sem.1 Review Presentation	Team	8	100%
5.2	Review Sem.1 Project Doc.	Team	8	100%
5.5	Review Sem.1 Progress with Clients	Team	4	100%
5.3	Complete Comprehensive Test Plans			
5.3.1	Write Feature Test Coverage Plan	Team	8	70%
5.3.2	Upload and communicate test plan	Derek	1	100%
5.4	Hand off Meeting with Oregon Team	Derek	1	100%
5.6	Review Final Project Doc.	Team	8	100%
5.7	Complete Final Review Presentation	Team	8	100%
5.8	Finalize Final Review Poster	Team	4	100%
5.9	Final Project Review with Clients	Team	4	100%

Appendix D - Team

1. Team Members

Derek Brandt, Tyler Orman, Aryan Rao, Israel Sanchez-Tellez, Danh Hoang

2. Required Skills for Project

The frontend interface of the Administration Portal is written using Angular and Typescript.
The backend uses the technologies Express and MySql to perform tasks.
The overall goal of DigiClips is to be able to host this project on AWS Lightsail.

3. Skill Sets Covered by Team

For the front end, Aryan has experience using the Angular framework.
Also, Israel has used Typescript in a previous Co-op for the front end.
The whole team has experience with MySql.
Aryan, Derek, and Tyler have brief experience with Express.
The part where the whole team lacks experience is with AWS Lightsail.

4. Project Management Style Adopted by Team

Overall, we use a hybrid approach to project management. For design, we chose the Waterfall style. This is because the project requirements must be understood before any development or design can be completed. For the implementation phase, we will switch to an agile style where the changes will be made into user stories that can be broken down into manageable tasks.

5. Initial Project Management Roles

Derek Brandt - Architecture Lead **TODO Need to elaborate on roles**
Tyler Orman - Security Lead
Aryan Rao - Programming Lead
Israel Sanchez Tellez - Client Interaction Lead

6. Team Contract

Team Members:

1) Tyler Orman
2) Derek Brandt
3) Israel Sanchez-Tellez
4) Aryan Rao
5) Danh Hoang

Team Procedures

1. Meeting Time and Location: Weekly Tuesday at 5 pm, Virtual.

2. Team Communication:

Discord will be the primary communication means via chat or voice call. Deadlines and scheduling will be done in the respective Discord channel.

3. Decision-Making Policy:

The primary decision-making policy will fall to a decision matrix. These types of decisions will be brought up during the weekly team meeting. Before the meeting, the individual is responsible for creating the categories of the decision matrix. The team will fill out the decision matrix to reach a group consensus. A majority rule can override, change, or remove any category in the decision matrix during the group meeting. Due to the length of the process, this method should be reserved for software architecture changes, team structure changes, and software structure changes.

Decisions on software implementation can be brought up during team meetings; however, they should be reserved for code reviews. Between two code reviewers and the submitter, a consensus on implementation should be reached by the majority rule. It should be escalated to a team meeting decision matrix if it cannot.

If any team member has a decision that does not fall under any previous category, it should be brought up to the team during team meetings. If the decision requires a decision matrix due to complexity, the person responsible must submit categories for analysis.

4. Record-Keeping Procedure:

At the beginning of each meeting, a group member will be chosen to write meeting notes. These meeting notes will be sent to the respective Discord channel at the end of each meeting. The responsibility of the recording will fall under a round-robin schedule, in the following order: Derek, Israel, Tyler, and Aryan. The schedule will move to the next person if a member misses a week. The person who missed will be required to record keep following week.

Participation Expectations

1. Meeting Expectations:

Each team member must attend all weekly meetings with the team and the client. During this, future deadlines will be discussed and agreed upon. If a member has hit a roadblock in, it is expected that they bring it up during the weekly team meeting. If a member has to miss a meeting for any reason, they likely inform the team; however, they only need to include why if repeatedly missing meetings.

2. Assignment Expectations:

Team members are expected to complete their assigned weekly tasks by the next meeting. If a team member is unable to complete their weekly task, the rest of the team should be promptly notified so that plans can be made to complete the task.

3. Response Expectations:

It is expected that each team member responds within 24 hours of a given message. Also, to ensure that communications are seen, it is expected that each member performs a daily Discord check.

4. Weekly Hours Expectations:

There is no maximum or minimum amount of time for each member to put into the project. However, it is expected that a team member finishes their weekly assigned tasks. If the weekly assigned tasks exceed 12 hours of total work, the member will not required to finish them. During the next team meeting, they should mention this excessive work so future weekly loads can be reduced.

Leadership

1. Team Roles:

- Derek - Project Manager / Architectural Lead - Organizing documentation, team meetings, and knowledge about overall project structure and organization.
- Israel - Client Interaction Lead - In charge of client and inter-team communication.
- Aryan - Programming Lead - Responsible for being knowledgeable in Angular and Typscript.
- Tyler - Security Lead - Identify security vulnerabilities and program.
- Danh - Programmer - Implement user stories and tests.

2. Support and Guiding Strategies:

The primary method to guide team members will be the weekly to-do list. This list will contain all tasks that need to be done and those that will be completed during the week. Researching technology is a valid to-do item. This will encourage team members to explore new technologies and obtain more efficient and elegant results before implementing solutions.

Each team member is expected to be adaptable and flexible. This means that team members are expected to help each other when hitting roadblocks, even if it means sacrificing meeting deadlines considered a lower priority. Team members are encouraged to participate in an open dialog during team meetings and throughout the week about roadblocks. Finally, all code will be subject to code reviews to allow members to learn about others' work and provide input.

3. Recognizing Contributions:

Each team member will report their accomplished tasks during the weekly meeting. This will be noted in the to-do as a visual representation of the work completed. Secondly, accomplishments will expected to be notarized in each progress report submitted.

Collaboration and Inclusion

1. Team Skills and Expertise:

- I (Israel) completed a Software Developer Co-op last year, where I worked on frontend code and the installer for the application. The frontend code consisted of Javascript and React components. I used the React Testing Library to test the frontend code. For the installer, I injected a dependency check using Wix# (WixSharp) for certain .NET Runtimes needed to run the application. This semester, I'm also doing a part-time co-op with the same company to continue gaining real-world experience.
- I (Derek) am a Software Engineer. I have experience writing embedded software in C++ and using Gmock, a testing framework that I have developed working at *John Deere*. This is my current job as a part-time student there. I have experience using Python machine-learning libraries such as Pytorch and Tensorflow in conjunction with GymAPI. I have experience programming games and game AI in Unity using C#. Finally, I have experience with backend and frontend development for web apps and Android apps that I have obtained through Iowa State courses.
- I (Tyler Orman) am a Cybersecurity Engineer. I have experience with networking as well as various programming languages, including C, Python, Java, and a bit of Javascript, HTML, and PHP. Through courses here at Iowa State, I have gained experience with Backend development and have also built various applications for different jobs. I currently work part-time for Iowa State as an intern with Hunter Strategy.
- I (Aryan) bring some hands-on experience in software development to the team. With a strong background in coding and problem-solving, I have honed my skills through various internships and work experience. This experience not only provided exposure to diverse technological landscapes but also fostered adaptability and a keen understanding of industry best practices. We, as a team, will be ensuring that we stay at the forefront of cutting-edge software development.
- I (Danh) am a Software Engineer I have experience from my internship with frontend site development and testing. I also have backend experience with some courses that I have taken at Iowa State. Some of the programming languages I know are Java, C, C++, Java Script and HTML. I also have experience with cloud somputing and experience with using databases like MySQL.

2. Team Encouragement and Support Strategies:
The primary strategy the team will use for encouraging and supporting contributions and ideas will be to have open communications. This means that as a team, we will hold meetings where any idea or perspective is valuable and will be heard by all. The team has a Discord server where chats and virtual meetings will occur.

Since DigiClips uses its own GitHub repository, the team will conduct code reviews. By doing code reviews, not only will the team ensure that minimal amounts of bugs reach production, but it will also allow team members to provide feedback to other team members.

Another strategy that will not be enforced but is highly encouraged is for team members to conduct pair programming sessions. These sessions include two or more team members working on the same task simultaneously. Pair-programming sessions create the opportunity for knowledge sharing and ensure everyone is actively involved.

3. Collaboration and inclusion Strategies:
All team concerns will be addressed with respect. These concerns should be raised during weekly meetings or mentioned in the Discord. We will implement an Open-door Policy to encourage each team member to vocalize concerns immediately. To ensure team members are collaborating, the team will use the weekly to-do list as primary evidence of the contributions of each team member. Any concerns regarding under or over-contributing should be addressed during the weekly team members.

Goal-Setting, Planning, and Execution

1. Team Goals:
The most fundamental goal of the team is to complete the course. However, the primary goal is to work with the given team to create a successful deliverable for Digiclips. Other objects include gaining experience handling functional and non-functional requirements, working with a client to create a deliverable and experience learning unknown technologies in short periods.

2. Assigning Team and Individual Work:
During each weekly team meeting, the team will assess what the client's needs are for the upcoming and which tasks to prioritize. All tasks will be recorded in the to-do list. At the end of each weekly meeting, tasks will be divided up amongst team members and will be recorded as in progress until otherwise completed.

3. Keeping on Task:
During weekly meetings, we will utilize a chime when the team gets off task. If a team member is absent at a meeting, it is their responsibility to read the meeting notes and check the to-do list to see what needs to be done. Since tasks will be assigned individually, it will be the individual's responsibility to keep themselves on task during work hours and adhere to prior expectations on week completion.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
Infractions of the contract will be addressed directly with the person responsible. If the person has a reason for a breach of contract, I.e., family or personal issues, then the breach of contract will be understood. However, a nonvalid breach of contract will be addressed during the team's weekly meetings.

2. What will your team do if the infractions continue?
If the infractions continue, the infractions will be escalated to the TA, then our academic advisor, and finally the professors of the class.

Team Member Signatures

---

a) I participated in formulating the standards, roles, and procedures as stated in this contract.
b) I understand that I am obligated to abide by these terms and conditions.
c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

Signatures	Date Signed
1) Tyler Orman	01/30/2024
2) Derek Brandt	01/30/2024
3) Israel Sanchez-Tellez	01/29/2024

4) Aryan Rao	01/29/2024
5) Danh Hoang	09/20/2024