Performance Monitoring Header

Design Document

Team 29

Buildertrend

Dr. Andrew Miner

Blake Dunn - Communications

Christopher Feltz - Quality Assurance

Doriane Hesseng-Ndoutoume - Frontend Lead

Lewis Sheaffer - Testing Lead

Michael Andrews - Project Manager

Robert Wise - Design Lead

Zachary Current - Backend Lead

sdmay22-29@iastate.edu

https://sdmay22-29.sd.ece.iastate.edu

Revised: 12/5/2021

Executive Summary

Development Standards & Practices Used

- WCAG 2.1 - accessibility standards will ensure that all users, regardless of their disabilities, are able to access the test website [3]
- ISO/IEC 12207:2008 - sets up the guidelines for the supply, development, operation, maintenance and disposal of software products [1]
- ISO/IEC 29119-1:2015 - defines the standards by which software is tested [2]

Summary of Requirements

- Performance Monitoring Header
  - Ability to monitor
    - API Response Time
    - Number of DB calls
    - Time spent in the DB
    - Web vitals (stretch goal)
  - Built with Typescript React
  - Use Ant design components
  - Customizable (stretch goal)
- Sample Website
  - Ability to test header

Applicable Courses from Iowa State University Curriculum

- SE 329
- COM S 309
- SE 319
- ENGL 314

- COM S 363

New Skills/Knowledge acquired that was not taught in courses

- Amazon Web Services (DynamoDB, Lambda Functions, API Gateway, S3 Bucket)
- Service Workers
- Typescript, React
- GitHub CI/CD

Table of Contents

Table of Illustrations

1 Team

1.1 Team Members

Blake Dunn

Christopher Feltz

Doriane Hesseng-Ndoutoume

Lewis Sheaffer

Michael Andrews

Robert Wise

Zachary Current

1.2 Required Skill Sets for Your Project

React

Typescript

Backend of choice

API development

Database Management

1.3 Skill Sets covered by the Team

React/Javascript/Typescript: Everyone except Blake

Backend/AWS: Zach, Lewis, Blake, Rob

1.4 Project Management Style Adopted by the team

Waterfall - First Semester

Agile/Scrum master - Second semester

1.5 Initial Project Management Roles

Communications - Blake

Testing Lead - Lewis

Quality Assurance - Chris

Project Manager - Michael

Frontend Lead - Doriane

Backend Lead - Zach

Design Lead - Rob

2 Introduction

2.1 Problem Statement

There are many steps in a typical website software development lifecycle such as the development stage, code review stage, and testing stage. The identification of performance issues found during these steps requires restarting the process, which costs time and money. The sooner issues are found, the faster and cheaper it will be to fix those issues. Developers not only require a quick way to access and assess these issues, but it must be done throughout the development process.

2.2 Requirements & Constraints

Requirements:

- The header must include the following statistics:
    - API response time
        - Loading multiple API calls
            - Grouping API calls made within the last 5 seconds of each other before displaying them on a header
    - Number of database calls that occurred on the back end associated with each API call
    - Response time of each database call that occurred on the back end
    - Front end web vitals (stretch goal)
    - Other front end metrics (e.g. time to first draw) (stretch goal)
- Create a sample website to test the header with
- The header must be extensible to allow addition or subtraction of metrics
- Support pages that call more than one API
- Baseline monitoring (stretch)

Constraints:

- Use Typescript/React
- Use the Ant Design React component library
- Use Prettier to format code

- Use ESLint to identify and report on errors
- The project must be completed in 2 semesters

2.3 Engineering Standards

WCAG 2.1 - Accessibility standards will ensure that all users, regardless of their disabilities, are able to access the test website. We will use ANDI to ensure accessibility standards are met [3]

ISO/IEC 12207:2008 - Sets up the guidelines for the supply, development, operation, maintenance and disposal of software products [1]

ISO/IEC 29119-1:2015 - Defines the standards by which software is tested [2]

2.4 Intended Users and Uses

Intended Users:

Developers and quality assurance engineers will use this performance header actively during their daily jobs. Additionally, users of Buildertrend's application will indirectly benefit from this project because the application's quality should improve with developers using the performance header.

Uses:

The project will be used by developers to monitor the performance of the application during the development process. They will be able to see statistics such as API response time and the duration of database calls. Seeing abnormal values will be able to alert the developers that they need to make modifications to the current application implementation.

3 Project Plan

3.1 Project Management/Tracking Procedures

Project Management:

Our group will adopt a mixture of waterfall and agile project management strategies to complete our project. Performing organized, linear project planning during the beginning stages of the project will allow us to build a solid project foundation and allow implementation in an iterative manner to occur smoothly in the post-planning stages. The majority of architectural and design decisions will be derived from these early waterfall-like stages, but following an agile methodology during production implementation will give us the flexibility necessary to complete the project efficiently.

Tracking Procedures:

Our group is using Git for version control in conjunction with Github for detailed collaborative and tangible tasks. Discord is our primary method of inner team communication and Microsoft Teams is our primary method of communication with our industry sponsor.

3.2 Task Decomposition

Header

- Design header using Ant Design
- Header prototype in React
- Header skeleton
- Generic statistic component
- Presentation layers for individual statistics
- Integrate header into test site

API Response Time Monitor

- Set up Service Worker to monitor HTTP traffic
- Record response times for one API call
- Record response times for multiple API calls

Database Time and Database Calls Monitor

- Integrate database for test queries
- Record database statistics from API headers

Testing Website

- General API framework setup via AWS
- Testing website skeleton/formatting
- Single requests (PUT, GET, POST, DELETE) + front end component
- Send X requests + front end component
- Send requests at Y frequency + front end component
- Varying request / response size + front end component
- Specify request response time + front end component
- External API calls + front end component

Header customization by end-user (Stretch)

- Settings storage
- Add / remove statistics function
- Reorder statistics function
- Aesthetic customization

Frontend performance metrics (Stretch)

- Monitor UI calls
- Log previous query times into a database + database setup
- Compare active query times with database data
- Custom front end metrics component

Baseline Monitoring for various performance metrics (Stretch)

- Log previous statistics in database + database setup
- Compare current statistics with database data
- Implement graphs on each component

Testing

- Setup testing frameworks
- Unit tests
- Integration tests
- Front end regression tests

CI/CD

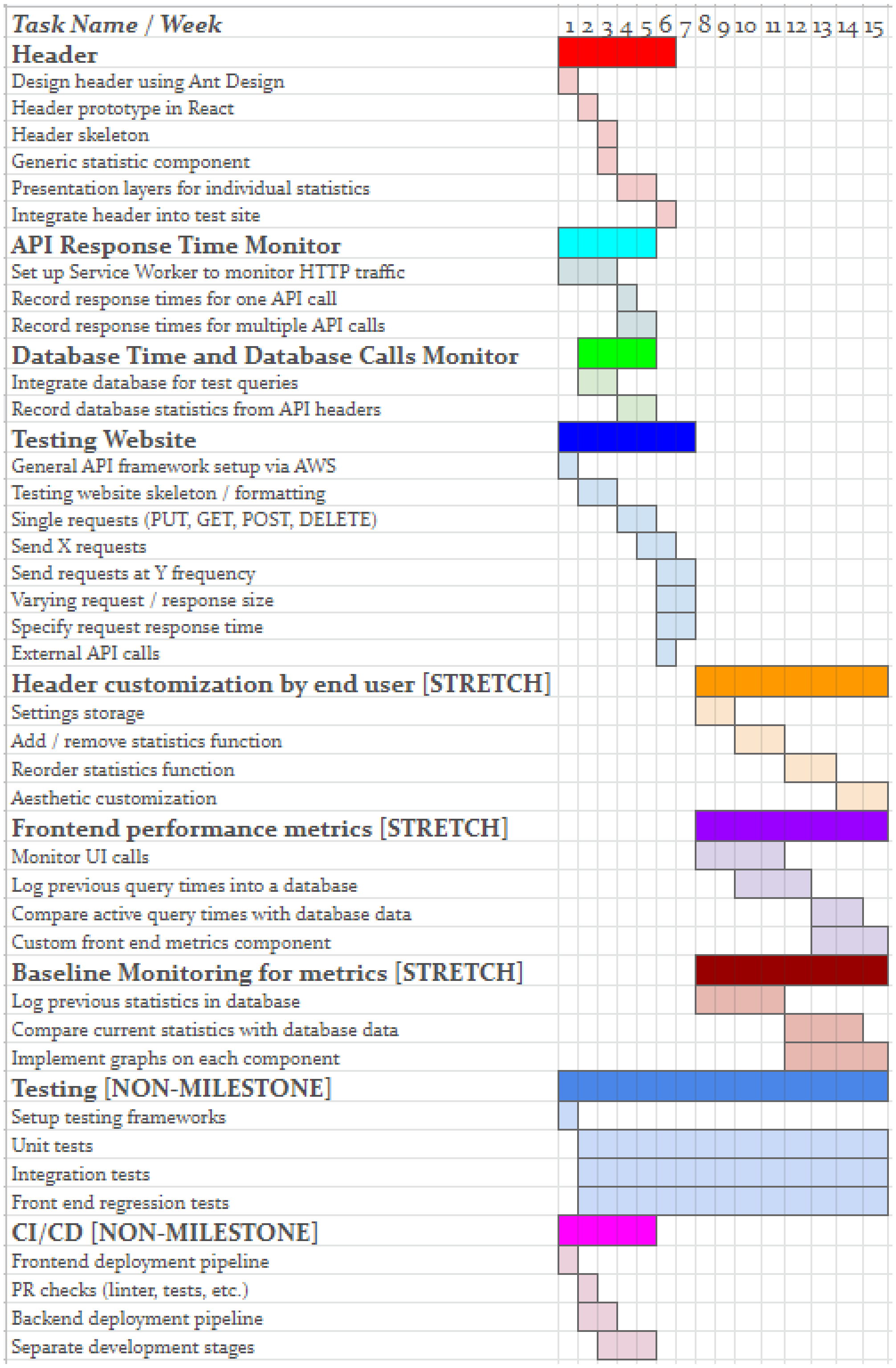
- Frontend deployment pipeline
- PR checks (linter, tests, etc.)
- Backend deployment pipeline
- Separate development stages

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Our progress is based purely on task completion, so progress towards a milestone will be based on the completion of sub-tasks. Only project deliverables are being considered milestones.

1. **Header**
   - 1.1. Header design / prototype
   - 1.2. Header skeleton in React
   - 1.3. Generic statistic component in React
   - 1.4. Presentation layers for individual statistics
2. **API Response Time Monitor**
   - 2.1. Record response times for one API call
   - 2.2. Record response times for multiple API calls
3. **Database Time and Database Calls Monitor**
   - 3.1. Record database statistics from API headers
4. **Testing Website**
   - 4.1. Single requests (PUT, GET, POST, DELETE)
   - 4.2. Send X requests
   - 4.3. Send requests at Y frequency
   - 4.4. Varying request/response size
   - 4.5. Specify request response time
   - 4.6. External API calls
5. **Header customization by end-user (Stretch)**
   - 5.1. Add / remove statistics function
   - 5.2. Reorder statistics function
   - 5.3. Aesthetic customization
6. **Frontend performance metrics (Stretch)**
   - 6.1. Compare active query times with database data
   - 6.2. Custom front end metrics component
7. **Baseline Monitoring for various performance metrics (Stretch)**
   - 7.1. Compare current statistics with database data
   - 7.2. Implement graphs on each component

3.4 Project Timeline/Schedule



Task Name / Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header	■	■	■	■	■	■									
Design header using Ant Design	■														
Header prototype in React		■													
Header skeleton			■												
Generic statistic component			■												
Presentation layers for individual statistics				■	■										
Integrate header into test site						■									
API Response Time Monitor	■	■	■	■	■										
Set up Service Worker to monitor HTTP traffic	■	■	■												
Record response times for one API call				■											
Record response times for multiple API calls				■	■										
Database Time and Database Calls Monitor		■	■	■	■										
Integrate database for test queries		■	■												
Record database statistics from API headers				■	■										
Testing Website	■	■	■	■	■	■	■								
General API framework setup via AWS	■														
Testing website skeleton / formatting		■	■												
Single requests (PUT, GET, POST, DELETE)				■	■										
Send X requests					■	■									
Send requests at Y frequency						■	■								
Varying request / response size						■	■								
Specify request response time						■	■								
External API calls						■									
Header customization by end user [STRETCH]								■	■	■	■	■	■	■	■
Settings storage								■	■						
Add / remove statistics function										■	■				
Reorder statistics function												■	■		
Aesthetic customization														■	■
Frontend performance metrics [STRETCH]								■	■	■	■	■	■	■	■
Monitor UI calls								■	■	■	■				
Log previous query times into a database										■	■	■			
Compare active query times with database data													■	■	
Custom front end metrics component													■	■	■
Baseline Monitoring for metrics [STRETCH]								■	■	■	■	■	■	■	■
Log previous statistics in database								■	■	■	■				
Compare current statistics with database data												■	■	■	
Implement graphs on each component												■	■	■	■
Testing [NON-MILESTONE]	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Setup testing frameworks	■														
Unit tests		■	■	■	■	■	■	■	■	■	■	■	■	■	■
Integration tests		■	■	■	■	■	■	■	■	■	■	■	■	■	■
Front end regression tests		■	■	■	■	■	■	■	■	■	■	■	■	■	■
CI/CD [NON-MILESTONE]	■	■	■	■	■										
Frontend deployment pipeline	■														
PR checks (linter, tests, etc.)		■													
Backend deployment pipeline		■	■												
Separate development stages			■	■	■										

Figure 3.4: Gantt Chart

3.5 Risks And Risk Management/Mitigation

Risk	Probability	Mitigation Plan
Header does not easily integrate into Buildertrend's website (Milestone 1)	0.3	Proactively try and use the same technologies that the client is using while developing the product. Write modular code for any debugging, use Prettier and ESLint to meet client standards.
Test website's API calls do not accurately mock Buildertrend's website functionality (Milestone 2)	0.5	Communicate with stakeholders about the format of API calls.
The database call times are inaccurate. (Milestone 3)	0.1	Use a preexisting performance monitor to compare our metrics to. Such as Datadog's end-to-end Modern Application Performance Monitor.
Test website has no place to integrate API calls (Milestone 4)	0.05	
There is not enough time to get front end metrics in javascript. (Milestone 5)	0.6	This is a stretch goal. If the risk increases toward the end, we will focus efforts on one or two metrics we would like to integrate.
Performance baseline monitoring contains sensitive information about the active use (Milestone 6)	0.1	Any information that is to be censored will be identified by the client. We expect this to come up during acceptance testing. We could develop a default censor system to hide information from being displayed. Proactively develop censor tool.
We don't have enough data to compare against for a baseline comparison. (Milestone 6)	0.2	We can create dummy data or store already available public data in a database. This data could be used to stress test the header.
Header has too many additional features integrated, unwanted features included (Milestone 7)	0.2	Modularity of our code will help us retroactively remove features during acceptance testing. As a stretch goal we have user settings, we can create feature flags to hide features based on user preferences.

3.6 Personnel Effort Requirements

Header - 70 hours

- Design header using Ant Design - 10 hours
- Header prototype in React - 10 hours
- Header skeleton - 10 hours
- Generic statistic component - 10 hours
- Presentation layers for individual statistics - 20 hours
- Integrate header into test site - 10 hours

API Response Time Monitor - 60 Hours

- Set up Service Worker to monitor HTTP traffic - 30 hours
- Track/record response times for one API call - 10 hours
- Track/record response times for multiple API calls - 20 hours

Database Time and Database Calls Monitor - 40 hours

- Integrate database for test queries - 20 hours
- Track/record database statistics from API headers - 20 hours

Testing Website - 120 hours

- General API framework setup via AWS - 10 hours
- Testing website skeleton/formatting - 20 hours
- Single requests (PUT, GET, POST, DELETE) + front end component - 15 hours
- Send X requests + front end component - 15 hours
- Send requests at Y frequency + front end component - 15 hours
- Varying request/response size + front end component - 20 hours
- Specify request response time + front end component - 15 hours
- External API calls + front end component - 10 hours

Header customization by end user (Stretch) - 100 hours

- Settings storage - 15 hours
- Add/remove statistics function - 20 hours
- Reorder statistics function - 15 hours
- Aesthetic customization - 20 hours

Frontend performance metrics (Stretch) - 120 hours

- Monitor UI calls - 40 hours
- Log previous query times into a database + database setup - 30 hours
- Compare active query times with database data - 20 hours
- Custom front end metrics component - 30 hours

Baseline Monitoring for various performance metrics (Stretch) - 110 hours

- Log previous statistics in database + database setup - 40 hours
- Compare current statistics with database data - 30 hours
- Implement graphs on each component - 40 hours

Testing - 130 hours

- Setup testing frameworks - 20 hours
- Unit tests - 40 hours
- Integration tests - 40 hours

- Front end regression tests - 30 hours

CI/CD - 90 hours

- Frontend deployment pipeline - 20 hours
- PR checks (linter, tests, etc.) - 10 hours
- Backend deployment pipeline - 20 hours
- Separate development stages - 40 hours

3.7 Other Resource Requirements

- A free tier of AWS that will be used to host our full-system application and manage the database that will store necessary application data
- VSCode will be used to write the code
- GitHub repository provided by Buildertrend

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	The performance header application must be accessible to all developers utilizing the tool. The header must be usable for individuals who use screen readers.	Public health, safety, and welfare
Global, cultural, and social	We will ensure that the application will be easily accessible to any Buildertrend developers who need such a tool for their development workflow.	Global, cultural, and social
Environmental	Our full-system application will use AWS for server hosting to allow for proper proof of concept before Buildertend decides to consume our repo to use our performance header developer dependency. AWS is more efficient as you only use computation when called while taking advantage of AWS's economy of scale.	Environmental
Economic	Inaccurate performance reporting during development could potentially result in Buildertrend's application performing poorly in production. This may ultimately result in loss of business for the company.	Economic

4.1.2 User Needs

Buildertrend developers need a convenient, informative method to monitor their application's performance across the stack, as such a solution will enable them to be aware of performance issues early on in the development lifecycle.

Buildertrend non-developers (e.g. project managers) also need an accessible way to monitor their application's performance because they might lack the expertise to examine other metrics through tools intended for experienced developers. While the primary long-term benefit of using the performance header will be reduction in code reaching production environments that negatively impacts performance, the performance header will also be a very useful tool for Buildertrend to tackle existing performance flaws in their application that may have already reached a production environment.

4.1.3 Prior Work/Solutions

Sumologic: https://www.sumologic.com/

Advantages:

- Provides detailed performance tracking metrics for cloud applications/infrastructures

Shortcomings:

- Professional license needed for enterprise-level service
- Metrics only would provide performance information regarding the backend server. Does not provide any performance tracking for frontend components
- The dashboard is only accessible through their website. This will limit its usability as a tool that provides immediate information for frontend development

Stage Monitor: https://www.stagemonitor.org/

Advantages:

- Open Source (Free to use)
- Provides detailed performance tracking for both frontend and backend applications/servers
- Metrics can be displayed in a frontend application using a provided widget

Shortcomings:

- Cannot access/display database query metrics
- Integration between Buildertrend's frontend and backend does not pass security constraints

LogRocket: https://logrocket.com/

Advantages

- Supports very detailed front end performance and use analysis for a variety of front end frameworks
- Detailed error logging and issue prioritization/categorization
- Alerts for problematic events and integration with external services such as Jira, Splunk, etc.

Shortcomings

- Professional license needed for enterprise-level service
- Large amount of analysis and metrics may be unnecessary overhead for some projects
- Integration with database response time is not directly provided

Browser Dev Tools: (e.g. CTRL+SHIFT+i in Google's Chrome browser)

Advantages

- In-house implementation not needed
- Available on every major browser
- Many front end performance metrics can be easily tracked

Shortcomings

- Not intuitive for non-developers; likely need training/development experience to use efficiently
- Not necessarily easy to monitor while performing large amounts of interaction on a site (browser dev tools can take up a large amount of screen real estate when viewing large amounts of data)

4.1.4 Technical Complexity

1. To test the limitations and accuracy of this performance monitoring header, a functional website and responsive API will need to be implemented. Since the header component will eventually be inserted into Buildertrend's website, our website must also replicate the functional complexity and the range.

2. The majority of the current solutions/products described above only provide performance tracking for one layer of a website's technology stack (UI performance tracking, Analytics for a backend server, ...). Where our project differs is that it provides performance metrics regarding the entire stack i n a visible header on the frontend website.

3. Outgoing HTTP request information is not readily available to elements within the DOM. Typical methods for monitoring such data are backend tracking/logging or browser dev tools, which are both tools that are either too external or visually obtrusive for quick data monitoring in real-time by Buildertrend developers.

4. Development of a sample website used for testing of the performance monitoring header should follow industry standards and provi de complexity present in modern web

applications (front end complexity in the DOM and back end API complexity) to provide a realistic standard to test against before our performance monitoring header is plugged into Buildertrend's application.

4.2 Design Exploration

4.2.1 Design Decisions

- Use AWS to host the sample website and back end
- Develop sample website as a testing tool for our performance header
- Use GitHub Actions for CI/CD integration
- Use service workers to intercept client network traffic (necessary for handling API-related performance data)

4.2.2 Ideation

The lotus chart shown below was used during the ideation of our design process. Using such a chart allowed us to brainstorm concepts and products that may be of use for the development of our performance header implementation.

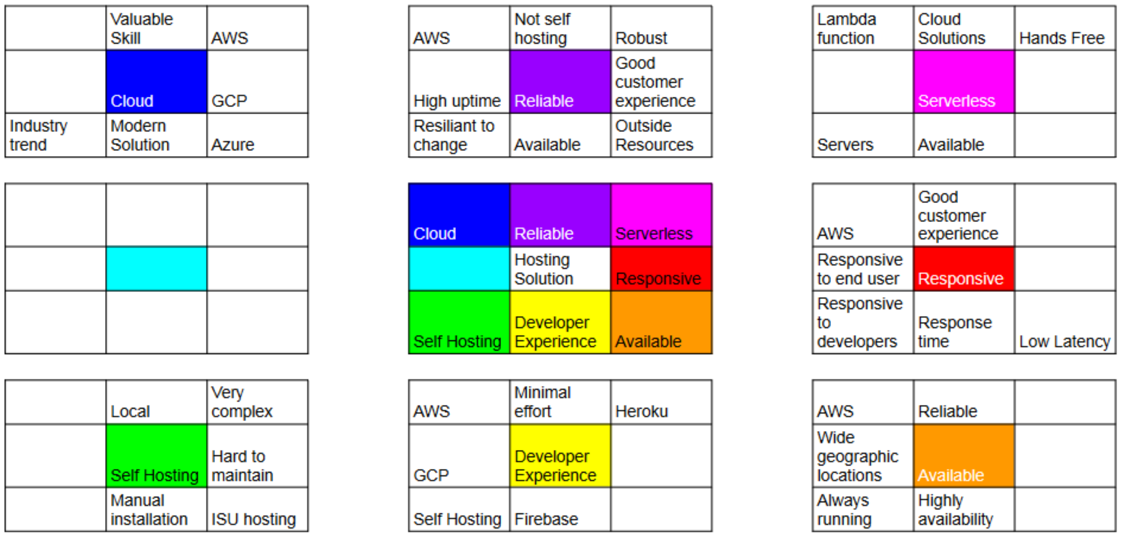


Figure 4.2.2: Ideation Lotus Chart

4.2.3 Decision-Making and Trade-Off

For our final choices, we wanted to choose something that we were familiar with as well as something that is similar to Buildertrend's current implementation. These were important considerations when we chose our hosting solution. In that case, most of the team members had some level of experience with AWS, Firebase, and ISU hosting. Buildertrend uses a traditional client-server implementation for their websites so this removed solutions such as Firebase and Heroku because they abstract the API calls into a library so you don't get the traditional HTTP response with headers and a body. Finally, we wanted something that had the ability to rapidly prototype because we did not want to be slowed down during our relatively short development time. Cloud solutions provide this. We chose to go with AWS because it fits all three criteria.

4.3 Proposed Design

So far, we have set up the backend and frontend hosting solutions. The backend setup involved creating a DynamoDB instance that is accessed through Lambda Functions and the Lambda Functions are exposed via the API Gateway. We have finished implementing a variety of APIs that are designed to test against edge cases that the performance header might encounter. These include APIs that have a large request size, a large response size, a long-running response, and a “typical” API that creates or queries a user.

The frontend hosting solution is a static S3 bucket. We have created an initial static performance monitoring header within Storybook and have started working on our sample website’s module that tests our user API. We were forced to do the prototyping in code instead of a design program because the Ant design components are locked behind a paywall. The prototype is in section 4.3.1.

To verify that we can intercept and read outgoing/incoming network traffic, we have also implemented a basic service worker prototype that listens to the fetch API calls of a simple React application. This prototype is capable of capturing the contents of the request when initially sent as well as capturing the contents of the corresponding response. Since the service worker exists outside the scope of the DOM, we have also verified that we still can send data/messages between the DOM and this service worker using the Broadcast API.

4.3.1 Design Visual and Description

<u>Back End Architecture</u>

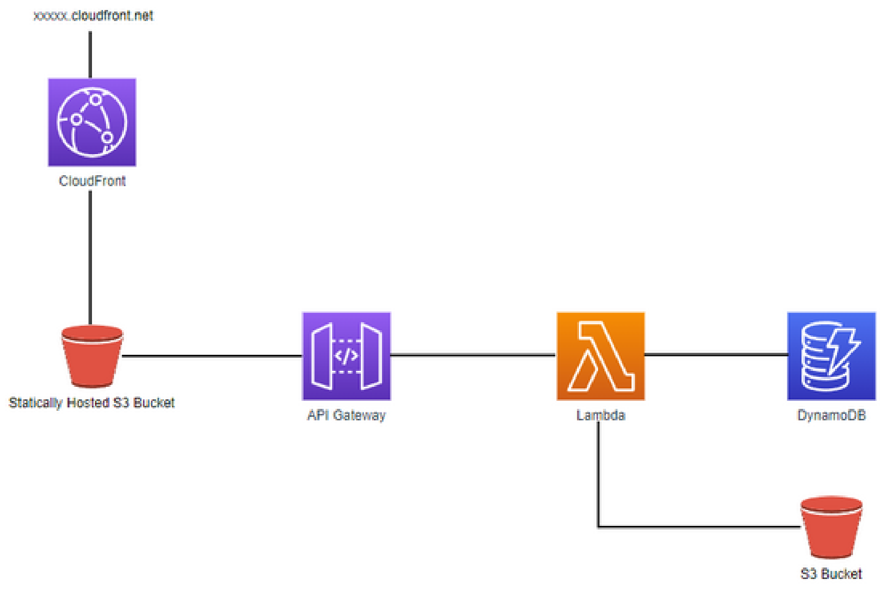


Fig 4.3.1.1: AWS services used for backend hosting, storage, and API interfacing

Figure 4.3.1.1 shows the Amazon Web Service architecture. For our solution we will be hosting our sample web application on a statically hosted s3 bucket. If we need to handle secure HTTP traffic we will utilize a cloudfront distribution to reroute HTTPS traffic. For our backend, we will use AWS API gateway to route API calls to Lambda functions to be processed. To store our sample data we will use DynamoDB for a non-relational database service. We will be using a secondary s3 bucket for sending and receiving files.

Front End Architecture

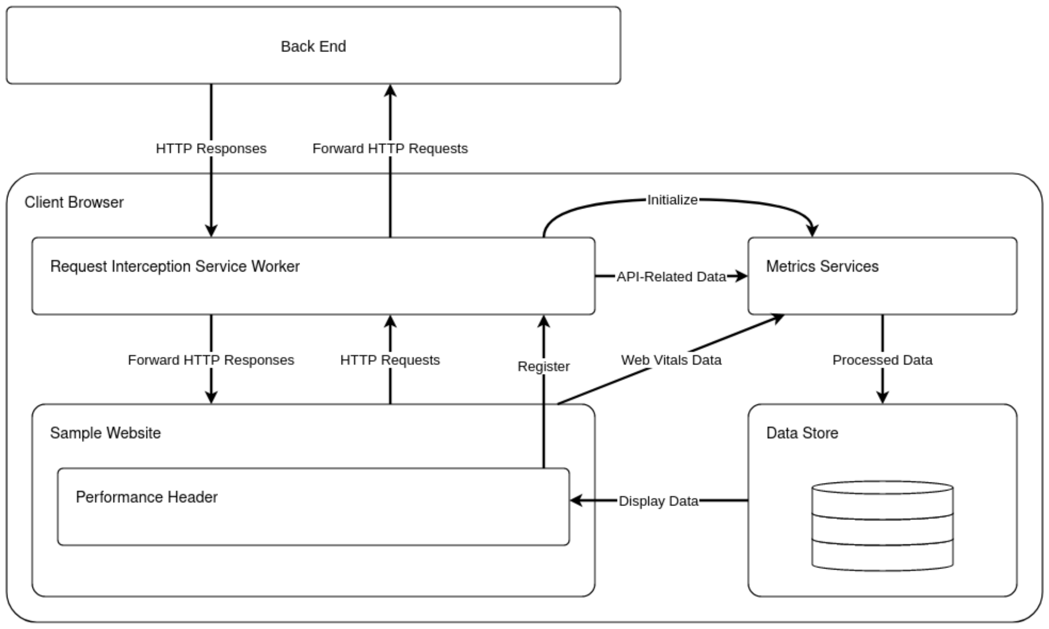


Figure 4.3.1.2: Front End System Architecture and Data Flow

The architecture of our application's front end consists of a few key components. Within a client's browser, a given website the performance header is plugged into will be available for a user (Buildertrend developer) to interact with. When the header is plugged in and the initialization process begins, a service worker is registered and services that run locally in a client's browser for processing performance data are initialized.

The service worker that is registered intercepts HTTP requests sent by the client through the website they are viewing. It then forwards those requests to the proper destination and intercepts the responses to those requests. Various pieces of data relevant to each API-related performance metric gathered through this interception process are then sent to designated metric services to be processed. Responses are then forwarded back to the execution path of the initial user request, creating no interference with the intended functionality of the website. The metric services will also pull data related to front end web vitals for the website. These metric services process the data and store it in a central local store within the browser associated with the website.

Finally, the performance header will pull data from the store (using the same consistent data models as the metrics services) to display necessary performance information to the user. Using a local data store will allow for simplified data persistence across sessions. Such a data store will likely come in handy when we attempt to perform baseline monitoring for various metrics as well (this is considered a stretch goal).

<u>UI Design</u>

The following UI components are the primary parts of the performance header that users will interact with according to the required goals we plan to meet.

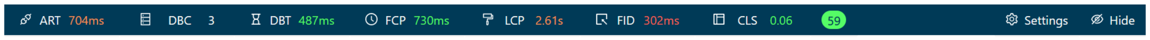
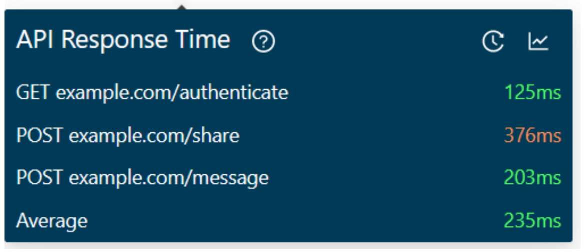

ART 704ms DBC 3 DBT 487ms FCP 730ms LCP 2.61s FID 302ms CLS 0.06 59 Settings Hide

Fig 4.3.1.3: The entire performance header that will be visible to developers when plugged into a website

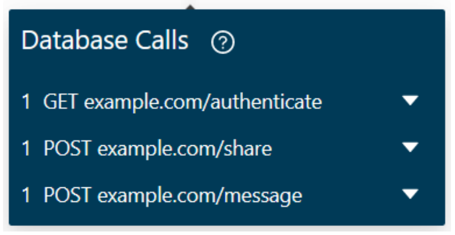
The entire performance monitoring header will display several API-related metrics and frontend web vitals metrics in a format that is efficient to obtain small amounts of recent data from. If users want to obtain more detailed data from recent times or other forms of data such as historical data, accessing the tooltips for each metric will be necessary.


API Response Time

GET example.com/authenticate	125ms
POST example.com/share	376ms
POST example.com/message	203ms
Average	235ms

Figure 4.3.1.4: The tooltip that will be visible to users when hovering over the API response time metric in the header

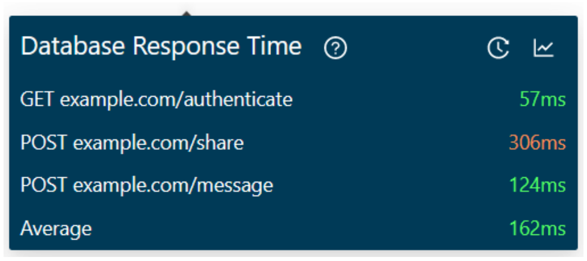
Figure 4.3.1.4 displays the tooltip related to API response time. Users will be able to see API response times associated with requests that occurred in the previous 5 seconds and larger amounts of historical data will be able to be viewed in list and chart formats by clicking the buttons at the top right corner of the tooltip.


Database Calls

1 GET example.com/authenticate
1 POST example.com/share
1 POST example.com/message

Figure 4.3.1.5: The tooltip that will be visible to users when hovering over the database calls metric in the header

Figure 4.3.1.5 displays the tooltip related to database calls. Users will be able to see the number of database calls associated with requests that occurred in the previous 5 seconds (this data is returned by the back end in request responses since such data is isolated to back end functionality) and greater details about such database calls will be accessible through expansion of the list items.

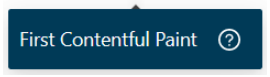


Database Response Time

GET example.com/authenticate	57ms
POST example.com/share	306ms
POST example.com/message	124ms
Average	162ms

Figure 4.3.1.6: The tooltip that will be visible to users when hovering over the database response time metric in the header

Figure 4.3.1.6 displays the tooltip related to database response time. Users will be able to see database response times (returned by the back end) associated with requests that occurred in the previous 5 seconds and larger amounts of historical data will be able to be viewed in list and chart formats by clicking the buttons at the top right corner of the tooltip.



First Contentful Paint

Figure 4.3.1.7: The general type of tooltip that will be visible to users when hovering over various front end web vital metrics in the header (first contentful print shown as an example metric)

Figure 4.3.1.7 displays what tooltips related to the several frontend web vital metrics will look like. Users will be able to click the question mark buttons to view explanations of the metrics to confirm understanding (these metrics tend to be single data values that are derived from a point in time, so we are planning to keep these tooltips simple for now).

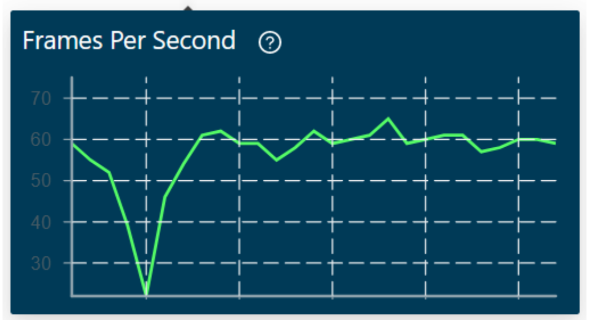


Figure 4.3.1.8: The tooltip that will be visible to users when hovering over the frames per second metric in the header

Figure 4.3.1.8 displays the tooltip related to frames per second. Users will be able to see a line chart that displays the frames per second metric across time in order to see spikes in dropped frames.

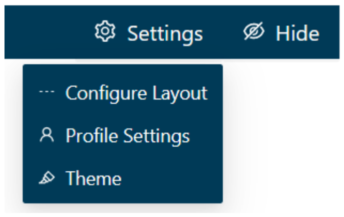


Figure 4.3.1.9: The menu that will be visible to users when hovering over the settings button in the header

Figure 4.3.1.9 displays the settings menu users will be able to interact with to configure the performance header in various ways. Users will be able to configure the layout of the metrics in the header, change settings that persist with their profile (such as settings related to data storage), and change the theme of the header through this menu.

4.3.2 Functionality

Our current design satisfies all of the functional requirements that were laid out by Buildertrend. The requirements that we are satisfying includes the following:

- Individual query metrics
- Summation of query performance from previous 5 seconds
- Query impact on the performance of the front end

Our group has taken the initiative of already updating the front end design. You can see the visual improvements from the original requirements that they had laid out. We have also elaborated on additional features such as a tooltip for the header, as well as graphs for displaying data analytics,

4.3.3 Areas of Concern and Development

Since we are not given access to Buildertrend's website code, our sample website has the potential to stray from the exact specifications of Buildertrend's site. This means that our monitoring header won't have the exact same test environment as the production environment, which could cause unforeseen difficulties when integrated into the actual site. For example, the DB time for an API call is apparently included in the response header for Buildertrend's website; we'd need to ask Buildertrend for more details on this to ensure our APIs work similarly. In general, our plan is to ensure our sample website covers all general website functionality, then conferring with Buildertrend for any unique aspects.

The client would like to be able to easily add and remove metrics from the header. While ensuring that more statistics can fit into the header should be trivial, integrating unique statistics is not. We'll need to implement the main statistics component such that it can be extended to perform unique calculations based on any new statistic. This should be possible by following basic OOP/design patterns. However, that doesn't cover the fact that our statistics have been purely numerical at this point. We'd need to communicate with our client to check whether non-numerical (e.g. true/false, string, range, something graphical) components might be included in the future, as this could require further abstraction in our design.

4.4 Technology Considerations

Highlight the strengths, weaknesses, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

- Ant Design:
    - Strength: UI design and development is expedited through use this library/design system; Ant Design contains a fairly large variety of UI components useful for many applications
    - Weakness: Using prebuilt UI component libraries can lead to unnecessary overhead if certain encapsulated features of UI components are not used in an application; prebuilt UI libraries may not provide all necessary components for a specific application and may lead to difficulties if in-house design/styling must occur to create “matching” custom components
    - Alternatives: Material UI
- React/Typescript:
    - Strength: Expedited development of complex front end applications, reusable component paradigm, simple dependency management, strong typing (unlike vanilla JavaScript)
    - Weakness: Lack of strict separation between DOM elements and functional logic could be considered an antipattern by some developers
    - Alternatives: Angular, Vue.js
- AWS:
    - Strength: Cloud providers eliminate the need for much of the manual work needed to set up infrastructure. The process of requesting a server and then installing all of the required software for this project was shrunk to a few simple clicks in the AWS console
    - Weakness: Cloud hosting can easily become extremely expensive when running the application in production because you trade convenience for cost. Additionally, all of the configurations that we did in the AWS console are specific to AWS. Moving to another hosting option would require redoing all of the configurations
    - Alternatives: Google Cloud, Microsoft Azure, ISU on-premise hosting
- Storybook:
    - Strength: Provides an organized, isolated React component development environment that encourages strong modularization and documentation practices
    - Weakness: Running a separate local application like Storybook to simply develop isolated React components could be viewed as unnecessary overhead for small projects
    - Alternatives: React Sketch.app

4.5 Design Analysis

So far, our design has not fully been implemented. Due to that, it is hard to determine if our design worked or not. However, based on what has been completed so far, we can say that our proposed design works for now. With our current implementation, we began two of our main requirements. These requirements are to create some APIs (Post, Get, Large request,...), which we did using AWS, and implement the testing website outline. They both work as expected for now.

At this point, we are satisfied with our design. However, we are aware that our design may be subject to change as we move forward with our implementation.

4.6 Design Plan

Our design plan focuses on three main parts: the header, the website, and the backend. They are in accordance with the requirements, the interfaces, and the different modules we are trying to accomplish in this project.

- Header: The performance header will be implemented using React JS and Typescript. Ant Design is a component library that will be used to design the header. A tool called Storybook will also be used to create the header. It allows us to build UI components in isolation.
- Website: The website will be hosted by AWS, specifically an S3 bucket. It will be used to create the different APIs that will be tracked in the performance header.
- Backend: AWS is also used to host the backend, more specifically DynamoDB, to store the metrics data and Lambda functions for the API gateway.

Below is a use-case diagram to better understand our design plan.

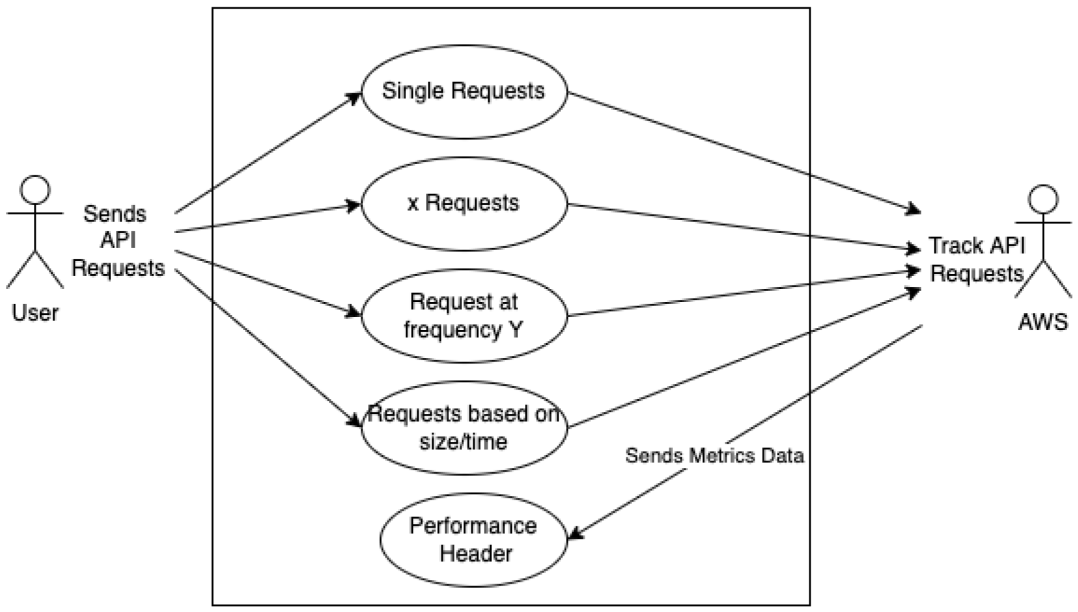


Figure 4.6.1: Use Case Diagram covering the requirements

5 Testing

5.1 Unit Testing

Unit testing will be done on a per-component basis which means that every React component that we make will have associated unit tests. These unit tests will be done using Jest. We will aim for approximately 80% code coverage to ensure that most cases are covered by tests. Jest will come preinstalled with Create React App.

5.2 Interface Testing

An open interface that we have in our design exists in the integration of AWS services. In particular, our AWS API Gateway with Lambda functions will access other AWS services as well. A great tool to test this interface is Postman. We will create a Postman collection that will run our API's and verify the correctness of the output. This collection can also be used for regression testing; as we make changes, we will need to test our old integrations as well.

5.3 Integration Testing

A major critical integration path based on our requirements will be the path between our backend database and the data being displayed on our header. As we know, integration testing assures that even if functions/components work properly individually, when combined together, they will still work as expected. One simple tool that can be used to test if the data generated in the back end are showing as expected in our header is unit testing. We will just combine the functions that cover these features and do a combined unit test to see if they're working properly together.

5.4 System Testing

This will use various combinations of the tests above (Our execution script for these tests in package.json can just reference specific groupings of the existing unit, interface, and integration tests). The preexisting tests (unit, interface, and integration) will be selected based on their relevance to larger functional requirements.

To accurately test and replicate the user's interactions with our component, it might also be a good idea to incorporate UI test automation in the browser. To do this, we will use a Javascript UI automation tool/package called Puppeteer. This tool easily integrates with Jest, so its tests can be run in conjunction with the other existing tests (as described in the previous sections above).

5.5 Regression Testing

As development progresses, unit, interface, and integration tests will be written to cover new functionality being added to the application. Having a strong understanding of all functionality present in each PR will be necessary to ensure full testing coverage. Running all existing tests, in addition to new tests added in each PR through an automated process (implemented in our CI/CD pipeline) and requiring 100% pass rate before merge is a viable strategy to prevent regressions across all levels of system functionality. Additionally, we must pay strong attention to PRs involving bug fixes and ensure proper tests are added that directly address the bugs being fixed to assure regressions causing the same bugs cannot occur again.

5.6 Acceptance Testing

For any functional requirements, we will be testing with the techniques mentioned in the previous sections. Of course, this is based purely on our understanding of the requirements, so we will still need to involve the clients for checking the functionality throughout. For non-functional requirements, we will need to involve the clients. We will of course check that each non-functional requirement is being included during implementation, but the final confirmation will need to be manual. As of now, we are planning to have meetings with our clients every other week to review the milestones that we have been able to cover since the last meeting. It is during this meeting that we should be able to confirm the functional and non-functional requirements have been covered for the milestone. Following the completion of our project, we will have a final review with our client as an overall acceptance test. For stretch goals, we will review each like a milestone in the following meetings.

5.7 Results

With the testing framework that we have laid out, we expect that we will have extensive coverage of our entire project. With unit testing being a common theme for every part of our system, we do not expect to have components that lack their required functionality. The compliance with the requirements will be met when the unit testing passes. Tests requiring automated user interaction will cover the functionality of multiple comprehensive parts of the system, and therefore require that all parts of our application stack are functioning correctly. Tests of this sort demonstrate the interdependence of the systems in our project and can be considered a novel proof of concept (acceptance testing by Buildertrend will be the final layer to establish proof of concept).

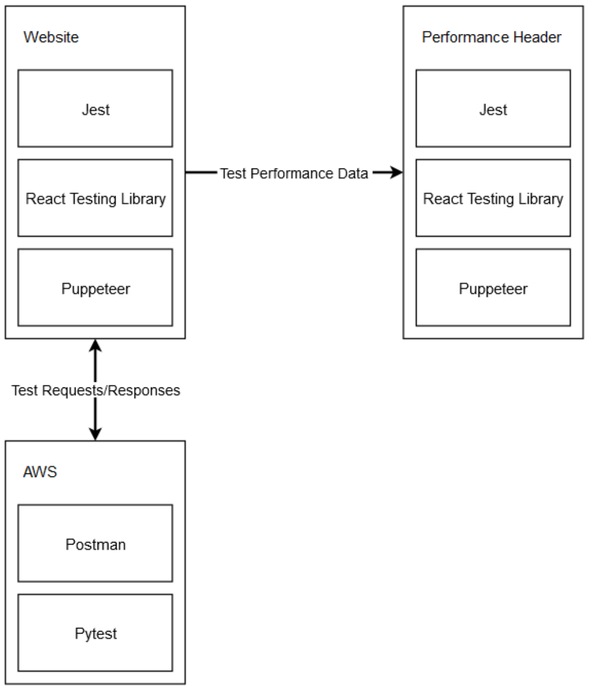


Figure 5.7.1: Summary of testing frameworks that'll be used to test our project

6 Implementation

For a smooth start to implementation, we have worked ahead to get much of the infrastructure in place before we officially start development. We've hosted our test website in an AWS S3 bucket, with a CI/CD pipeline in place to deploy changes to it. Additionally, we have a basic backend set up using AWS Lambda + API Gateway. Lastly, we drafted out a header with Ant Design and React, which we showed to our client to make sure it met their needs. All of these steps put us slightly ahead of our Gantt chart and ensure that we won't run into (as many) unexpected issues with our design choices.

In the next semester, we plan to work on the header, test website, and CI/CD concurrently. We're attacking the CI/CD work early, since we'd reap the benefits further down the road. Since the header and APIs aren't completely reliant on each other, we're hoping to split the team between them to get them finished within the first half of the semester. As soon as those are completed, we'll split the team between our three stretch goals, which are also relatively independent. Throughout the entire semester, we'll be meeting with our client bi-weekly to get our milestones reviewed. Additionally, we'll meet with them to discover more requirements, should we complete our stretch goals before the semester closes. On a final note, we plan to keep working on tests throughout the semester, whenever new functionalities are added.

7 Professionalism

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

Pick one of IEEE, ACM, or SE code of ethics. Add a column to Table 1 from the paper corresponding to the society-specific code of ethics selected above. State how it addresses each of the areas of seven professional responsibilities in the table. Briefly describe each entry added to the table in your own words. How does the IEEE, ACM, or SE code of ethics differ from the NSPE version for each area?

Area of responsibility	Definition	NSPE Canon	SE Code of Ethics
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	Principle 3 Product and Principle 4 Judgment Products/modifications will be developed with the highest professional standards. Strong documentation and testing is expected. Developers are expected to be competent and have experience with the projects they are working on. This differs from NSPE since there is a greater emphasis on following strong software development practices (documentation, testing, debugging, ...).
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	Principle 5: Management managers should be ethical in their management of teams and

			software projects NSPE focuses more on the financial aspects while the SE code of ethics focuses more on management styles
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	Principle 2: Client and Employer Software engineers should act in the best interest of the employer, client, and public Both the SE Code of Ethics and NPSE explicitly state that you should be fair and avoid deception in public statements. The SE Code of Ethics also holds a higher standard to the level of confidentiality to be used while communicating.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders	Hold paramount the safety, health, and welfare of the public.	Principle 8: Self Software engineers should continually improve their professional skills such as communication, reliability, documentation, etc. The NSPE code of ethics focuses on the engineers communicating with their employer/client. The SE code of ethics focuses more on the engineers' self-improvement.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	Principle 2: Client and Employer Software engineers should use the property of the client or employer in ways that are properly authorized with the consent of the employer or client's knowledge and consent. The NSPE is focused on concerns that may fall into the engineer's actions or status with third parties. This can be seen in the

			multiple rules pertaining to conflict of interests. The rules that are in the Software Engineering Code of Ethics are more focused on maintaining a client's reputation and keeping them informed on any developing changes.
Sustainability	Protect the environment and natural resources locally and globally.	Engineers should at all times strive to serve the public interest.	<u>Principle 1: Public</u> Software engineers should only approve of and develop software that does not result in harm to the environment or diminish the quality of life for the public. The notable difference between the NSPE and the SE code of ethics is the wording of "encouraged" in the NSPE ethics. This gives some leniency to the software that the engineers are writing and is therefore not as strict.
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.	<u>Principle 1: Public</u> Software engineers should operate in the best interest of the public, cooperate in efforts to address problems of public concern, and incorporate usability considerations for diverse audiences into their work. This is quite similar to NSPE, but the SE code of ethics tends to have a greater focus on preventing deception and reducing support of software that is knowingly impacting society in some negative way (not just avoid software but disclosure of said knowledge should be performed as well). There is also an added layer involving the inclusion of audiences and volunteering efforts in the SE code of ethics.

7.2 Project Specific Professional Responsibility Areas

For each of the professional responsibility areas in Table 1, discuss whether it applies in your project's professional context. Why yes or why not? How well is your team performing (High, Medium, Low, N/A) in each of the seven areas of professional responsibility, again in the context of your project. Justify.

Work Competence -

Applies: Yes - This product needs to be developed with the highest professional standards. To provide a sufficient product to the client we need to be competent and experienced in the tools to produce this product. We will also need to thoroughly document and test this product.

Performance: Medium - We have documented the design of the product sufficiently. However, due to the lack of implementation, we have not set up sufficient testing. Between the team members, we have sufficient knowledge and experience to complete this project.

Financial Responsibility -

Applies: Yes - The development of this product can be done with little to no investment. We also need to follow standard software security practices to ensure sensitive information such as usernames, passwords, and API keys are protected.

Performance: High - As a team, we have been researching ways to develop this product following best security practices as well as with minimal financial risk. To minimize financial risk we have decided to use AWS free tier subscription to set up the Proof of Concept for our product.

Communication Honesty -

Applies: Yes - As a team, we owe it to our client, our instructors, and ourselves to report our work truthfully and honestly.

Performance: High - We have maintained contact with our TA and have updated them on our progress throughout the design process. We have also maintained contact with our client and have kept them informed of our progress.

Health, Safety, Well-Being -

Applies: No - Our product does not have an effect on the well-being of our clients.

Performance: N/A - Not applicable to our product.

Property Ownership -

Applies: Yes - Any and all information we receive about the client should be used properly and with consent of the client.

Performance: High - We have signed the Intellectual Property Agreement as well as the Non-Disclosure Agreement with our client. Any and all products or solutions that are developed for our client is property of our client and will be treated as such. We have limited access to our client's resources to appease the principle of least privilege.

Sustainability -

Applies: Yes - As developers, we need to use the appropriate amount of resources for our project. Not doing so could contribute to the overall problem of over-consumption of energy.

Performance: High - We are utilizing efficient Amazon Web Services for computing and storage. For example we are using AWS Lambda for computation, which will use servers on an on-demand basis.

Social Responsibility -

Applies: No - The product we are developing is to solve an internal issue for our client. This product will have little to no benefit or disadvantage to the public.

Performance: N/A - Not applicable to our product.

7.3 Most Applicable Professional Responsibility Area

Considering our client meetings are 100% remote, it would be easy to be dishonest with our progress or withhold information from our clients. Due to this unique environment, the professional responsibility area of communication honesty is extremely important for our project. Without communication honesty, our clients would be unable to effectively critique our work, potentially causing us to produce a product inconsistent with their expectations. So far, our team has exhibited this professional responsibility area by informing the clients of our shortcomings. In one instance, we had a UI demo of our project built in React, but the code was a hastily made proof-of-concept. When demoing, we showed the client this demo but were sure to inform them that it would need to be rebuilt to improve the code quality. Our client was understanding of this, and it allowed them to temper their expectations to meet our current state.

8 Closing Material

8.1 Discussion

Our group has successfully designed the product. The requirements laid out by our client, Buildertrend, have been met with the design that we have created. The result of our project will be a complete end-to-end monitor for developers at Buildertrend to use. Our project will decrease the development time required as it will proactively be identifying problems for developers to solve. The sooner problems are identified, the less time will be wasted down the line.

8.2 Conclusion

Our group has a concrete design for what we are going to be doing. We have a plan on what we are going to be implementing, as well as when we are going to be implementing it. We have established our AWS account and created samples for each of the services we will be using.

On our front end, we have a mockup of the header, we are connected to and are sending requests to the back end API. We have set up settings in our config.json file in order to create a coding standard for any development being done. Our group has also set up Storybook in order to document any development that is taking place. Lastly, we have also begun to implement service workers to monitor any HTTP requests being sent.

On our backend, we have set up the different AWS components that have been discussed in this design document. We have yet to implement much of the functionality that will be necessary for our deliverable. We have successfully connected the frontend and backend, with successful queries and responses being logged.

8.3 References

[1] "ISO 12207," *Process Improvement with CMMI® v1.2 and ISO Standards*, pp. 1–357, 2008.

[2] S. Ali and T. Yue, "Formalizing the ISO/IEC/IEEE 29119 software testing standard," *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015.

[3] W. C. W. A. I. (WAI), "Web content accessibility guidelines (WCAG) overview," *Web Accessibility Initiative (WAI)*. [Online]. Available: https://www.w3.org/WAI/standards-guidelines/wcag/. [Accessed: 06-Dec-2021].

8.4 Appendices

N/A

8.4.1 Team Contract

Team Members:

1) Michael Andrews	2) Zachary Current
3) Blake Dunn	4) Christopher Feltz
5) Doriane Hesseng-Ndoutoume	6) Lewis Sheaffer
7) Robert Wise	

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

Purpose	Day/Time	Location	Medium
Weekly TA Meeting	Tuesdays 5pm	Coover TLA	Face-to-face
Bi-weekly Industry Meeting/Milestones	Every other Thursday 4pm	Microsoft Teams Meeting	Virtual
Weekly standup	Tuesdays 5:30 PM	Coover TLA	Face-to-face

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

We will use Discord to communicate. Everyone is expected to check the Discord at least twice per day to get any critical information.

3. Decision-making policy (e.g., consensus, majority vote):

When making decisions, we will take a vote and the proposal with the majority vote will be used.

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Chris will be taking the meeting minutes. Notes and other shared materials will be uploaded onto our shared Google Drive. The meeting minutes will be appended to the Meeting Meetings document.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

All members are expected to attend all meetings. Missing a meeting will require prior notice in the Discord.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Everyone is responsible for completing their allotted work prior to the due date. Group assignments must be submitted 24 hours prior to the canvas due date.

3. Expected level of communication with other team members:

Everyone is expected to participate in communication in a timely manner if directly contacted by another team member. If group communication is occurring, everyone is expected to follow discussions as necessary in order to stay updated with information relative to group member responsibilities.

4. Expected level of commitment to team decisions and tasks:

Everyone must follow the decisions made by the team. Team decisions will be made through democratic votes.

<u>Leadership</u>

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Communications - Blake

Testing Lead - Lewis

Quality Assurance - Chris

Project Manager- Michael

Frontend Lead - Zach

Backend Lead - Doriane

Design Lead - Rob

2. Strategies for supporting and guiding the work of all team members:

Weekly standup meetings to keep everyone on the same page for expectations. When team members are struggling, they can request to pair program or mob program with the rest of the team.

3. Strategies for recognizing the contributions of all team members:

The weekly standup meetings will be recorded via text on discord. This will show the weekly contribution that members have made.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

Chris - Frontend development with React. My unique perspective would be that I am the only member from Illinois.

Zachary - Frontend Development, including frameworks like React. Experience with performance monitoring. AWS development experience

Blake - Backend development, Infrastructure/system administration, automation, Android development

Doriane - Frontend and back end development experience including Reactjs, Spring Boot, Agile methodology, CI/CD, SQL

Michael - Full-stack development, modern JS/TS tech stacks including frameworks such as React, React Native, Express, etc, limited Google Cloud experience

Rob - Frontend stuffs (React, some Angular), some back end stuffs, random AWS tool experience, MongoDB, design doc experience, testing experience, some CI/CD stuff in GitLab, "unique" perspective as a music major I guess

Lewis: Full Stack development: Java, Javascript, React, HTML, CSS, Spring/Spring Boot, CI/CD, test automation

2. Strategies for encouraging and support contributions and ideas from all team members:

We have open discussions to let everyone know that they can speak. Any important decisions will be held in a council type discussion. Go around the table and let everyone say their piece.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

Members of the team should first consult Michael about any concerns and then he can bring them up with the rest of the team.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

Complete the design specifications thoroughly.

2. Strategies for planning and assigning individual and team work:

Assign team members with projects as they are given to us. Most work will be assigned to at least two members to ensure completion

3. Strategies for keeping on task:

Start assignments early and work through them periodically. Keeping other members informed of progress during assignments through weekly asynchronous updates.

<u>Consequences for Not Adhering to Team Contract</u>

1. How will you handle infractions of any of the obligations of this team contract?

When a team member does not follow the team contract, it is the responsibility of the team to notify the offending team member to stop.

2. What will your team do if the infractions continue?

If a team member continues to break the team contract, the team will bring up the infractions during the weekly meeting with the TA.

****

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.

1) Michael Andrews	DATE 9/16/2021
2) Zachary Current	DATE 9/16/2021
3) Blake Dunn	DATE 9/16/2021
4) Christopher Feltz	DATE 9/16/2021
5) Doriane Hesseng Ndoutoume	DATE 9/16/2021
6) Lewis Sheaffer	DATE 9/16/2021
7) Robert Wise	DATE 9/16/2021