WCM Tool Health Analyzer

JIC 4336

Michael Cao, Arun Murali, Justin Kamina, Chenming Feng, Manuel Morin

Client: SLB

Repository: https://github.com/JamminJustin/4336-WCMHealth

Table of Contents

List of Figures

Terminology

API – Application Programming Interface: Contract that defines how applications interact with each other

Back-end: The part of the application that cannot be accessed by the user.

BHA – Bottomhole Assembly: The lower portion of a drilling rig.

C# (Code-Behind): Handles the logic, event handling, and interaction with UI components defined in XAML.

Front-end: The part of the application the user interacts with.

LWD – Logging While Drilling: A technique of sending tools alongside a drill to gather data while drilling.

MWD – Measurement While Drilling: A technique that uses sensors to collect real time data about drilling rigs.

WCM – Well Construction Management: The management of the practice of drilling a hole into the ground and extracting water via a well.

WPF - Windows Presentation Foundation Framework: Provides a comprehensive set of application-development features. Part of .NET (Backend).

XAML - eXtensible Application Markup Language: Used for designing the user interface (UI). It provides a declarative syntax for building rich and responsive UI elements.w

Introduction

Background

In the rapidly evolving field of Well Construction Measurement (WCM), efficient and effective decision-making is crucial to maintaining productivity and minimizing downtime. Well construction management refers to the coordination and planning of the drilling of wells, and the management of the tools and processes that come with that. Field operations increasingly require quick and accurate assessments of the health status of bottomhole assembly (BHA) tools to determine whether they can be reused without servicing for the next job. This "go-no-go" decision is critical to ensuring safety, reliability, and cost-effectiveness in WCM activities.

Currently, the process of evaluating the health of BHA tools relies heavily on manual examination and interpretation of tool dump data, which is both tedious and time-consuming. This manual approach not only delays decision-making but also increases the risk of human error, potentially leading to costly delays, equipment failures, or suboptimal use of resources. As field activities continue to grow in volume and complexity, there is an urgent need for an automated, robust, and user-friendly solution to analyze tool dump data efficiently and consistently.

Our project aims to revolutionize the assessment of BHA tool health by providing an automated, scalable, and reliable platform for processing tool dump data. Our framework will enable field teams to quickly and accurately generate health reports, complete with "go-no-go" decisions, thereby reducing the time and effort required for data analysis and minimizing the risk of human error. By delivering a working app, source code, and comprehensive documentation, we will empower field operations to make faster, data-driven decisions, enhance tool management efficiency, and improve overall operational safety and performance in the WCM field.

Document Summary

The document begins by laying out a table of contents, followed by a list of figures documenting the location of sections and figures. Then, it discusses the background of the project, followed by this summary.

Next, it discusses both the static and dynamic system architecture of our project with figures. The *system architecture* section describes both the static and dynamic architecture of the system. It describes how components relate to each other and contains diagrams describing their relationships.

Then, the *Component Design* part analyzes the structure of the health analyzer. It breaks up the key components, splitting up the Local Storage, Main Page, Config Page, and Engine.

After that comes the *Data Design,* which discusses how data is handled throughout our application end-to-end from the formatting of the inputs to the way it's exported.

Finally, we have the UI Design portion, which details what the end-user sees and what a typical user flow looks like through different screens.

System Architecture

Introduction

In this document, we have provided a detailed description of our system architecture design. The two diagrams below represent the Static and the Dynamic system architecture diagrams respectively, which give a high-level description of the components of our application and their relationship. The main components are the User, their Local Storage, the Application – which contains the Interface and the Controller, the Engine, and the company's External Library. Both the diagrams explaining our application contain corresponding core components making it easier to understand the relationships between them.

Static Architecture

The static architecture of our Tool Health Analyzer gives a blueprint of the major components and their interactions without any circular dependencies. Our design follows a logical and hierarchical structure, ensuring a seamless flow of information through various layers of the application. A diagram for our static system architecture can be seen below:

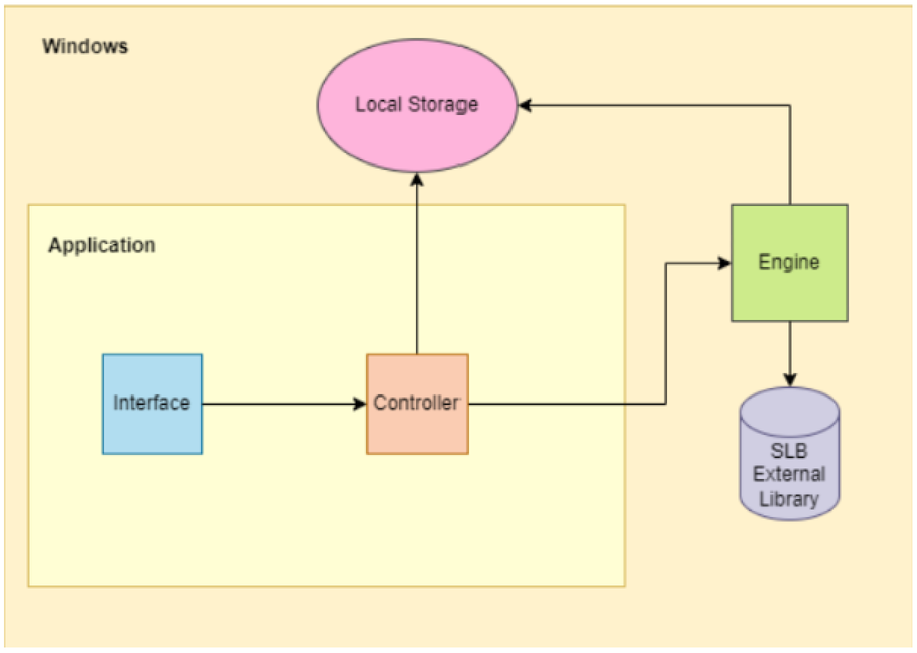


Figure 1: Static System Architecture

The Local Storage is a repository for all data files relevant to tool health stored on the user's device. These files serve as an input for analysis. The Local Storage retains the data until it is called upon by the

Engine for processing, ensuring that the user's information remains intact and accessible by the tool's operational cycle.

The Interface is the user's gateway into the application, presenting a responsive front-end that allows user interaction with the system. It captures user inputs, displays data, and communicates user commands to the Controller. This component is designed to be user-friendly because it allows complex operations to be translated into a simple and effective user experience.

The core of the application's logic is the Controller, which orchestrates the system operations. It retrieves instructions from the Interface and translates them into actionable tasks usable by Engine. During the report generation process, the Controller works with the Interface to gather configuration details, retrieves the data needed from Local Storage, and prompts the Engine to start the analytical processes.

The Engine is the analytical powerhouse, which lies outside of the application structure. It is responsible for processing input data and configuration settings to produce detailed reports. It operates in tandem with the application or as a standalone component when accessed through command line, providing flexibility in how the analysis is conducted according to SLB’s specifications.

The SLB External Library is part of the Engine's suite. It is a collection of specialized algorithms and code snippets that the Engine uses to perform calculations. These libraries improve the Engine's analytical capabilities, allowing it to generate accurate reports based on the configuration settings.

Dynamic Architecture

The dynamic architectural design highlighted below shows the system's operational flow and illustrates the interactions between its components during runtime. This sequence diagram clarifies how the user's actions trigger processes within the application and how these processes are handled internally to achieve the desired outcome.

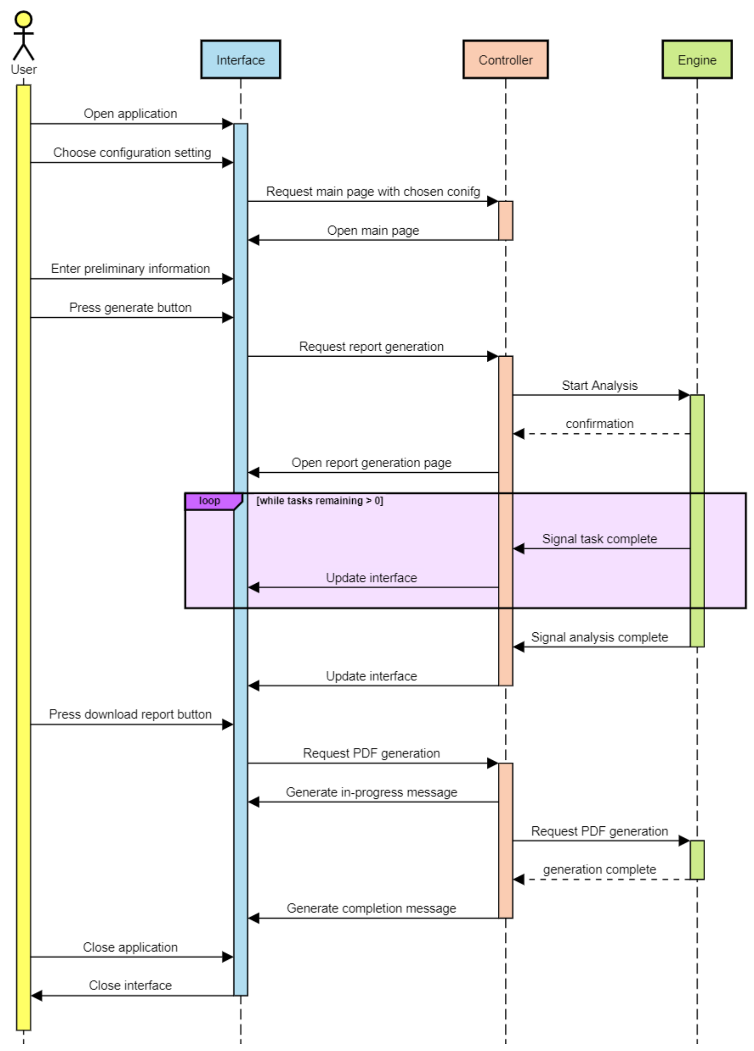


Figure 2: Dynamic System Architecture

Based on the diagram above, once the Tool Health Analyzer runs, the user starts interacting with the Interface, selecting the configuration settings that will dictate the analysis parameters. The Interface then communicates this selection to the Controller, which opens the main application page reflecting the chosen configuration. Here, the user provides preliminary information that allows the data analysis and report generation process to be modified according to their specific needs. Once the preliminaries are set, the user commands the system to generate a report by pressing its corresponding labeled button. This action sends a signal to the Controller, which acts as the central hub by conveying the signal as instructions to the Engine to start data analysis. The Engine acknowledges this command. It follows its internal logic and the defined configurations settings to begins processing the data. During this phase, the Engine continues communicating with the Controller, signaling the progress of the analysis. As the Engine parses the data, the Controller operates user interface updates, informing the user of the ongoing analysis with the Interface. After analysis is complete, the Engine notifies the Controller, triggering the Interface to display the report as available for review. If the user downloads a report, they interact with the Interface, sending a download request to the Controller. The Controller then instructs the Engine to generate a PDF version of this report. While the PDF is being prepared, the user is notified with an in-progress message. Upon completion, a message indicating the report is ready, and the user can then download the finalized PDF document. The process ends with the user exiting the application. This causes the Interface to close, thus ending the user session. This sequence of interactions demonstrates the system's responsiveness to user inputs, ensuring a smooth progression from initiation to conclusion for the data analysis. Overall, we made our dynamic interaction model from architectural choices, aimed at optimizing user experience with a responsive design, minimized wait times, and accurate data analysis process. The separation among the components gives a scalable system that adapts to future enhancements with minimal disruptions to our established workflow.

Component Design

Introduction

This section provides a detailed examination of the structural and runtime aspects of the Tool Health Analyzer. Expanding on the system architecture, this section defines the static and dynamic elements of the system and ensures a clear and precise understanding of the design at a closer level. By zooming in on key component interactions, this section establishes a strong connection between the vague high-level architectural concepts and their actual implementation, reinforcing the system's integrity and rationale.

Static Elements

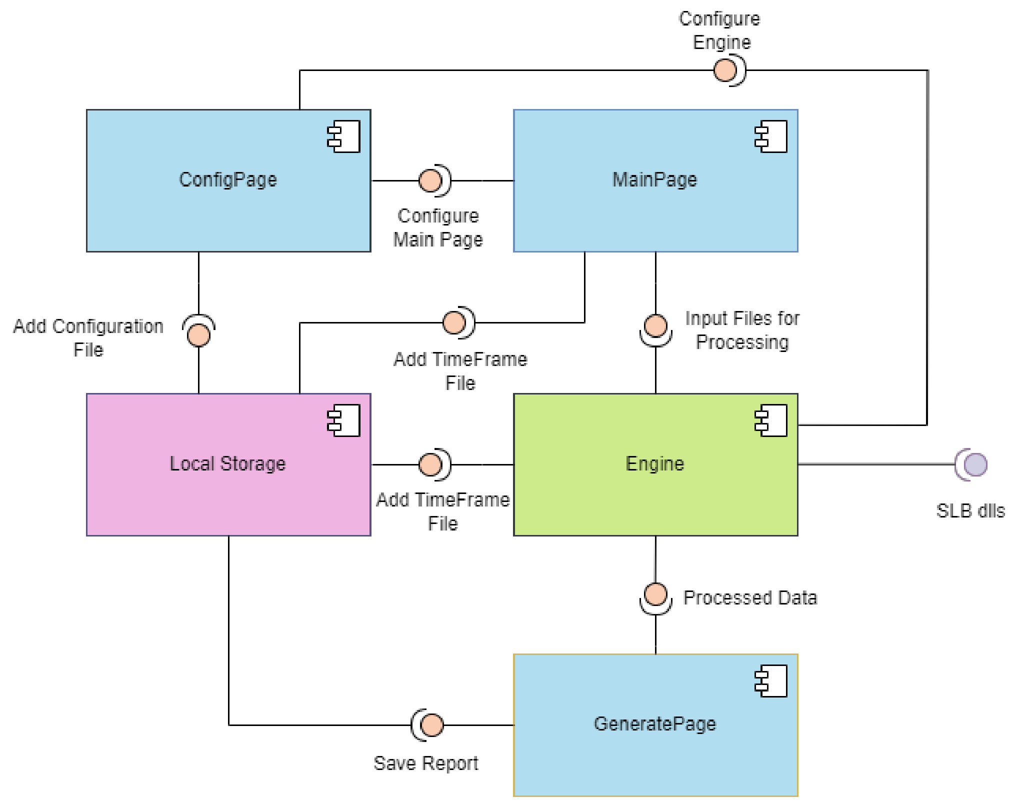


Figure 3: Static component diagram

The component diagram above represents the application's primary components and their interactions. The ConfigPage component is an interface responsible for specifying the layout of the application's MainPage interface, which is dependent on the currently loaded config file. It interfaces with Local Storage, where configuration files are located, and with the Engine, where it configures operational parameters, executes time frame file pre-processing and fault detection algorithms. The MainPage is where the user adds the required input files and any additional inputs. The Engine is the core processing unit of the application. It verifies input files and processes data received from the MainPage using SLB provided DLLs. The processed data is then given to the GeneratePage, which is responsible for displaying the report as PDF. Reports can be saved to Local storage for future viewing.

Dynamic Elements

Figure 3 below depicts an Interaction Overview Diagram of the process of loading a configuration file, selecting your preferences, and viewing the report. In the beginning, the user must input a specific configuration file. If the file is invalid, the system will backtrack and allow the user to submit a new one. If the file is valid, the system then loads the main page with the specific configuration properties defined in the file. The Interface communicates with the Controller to generate the new main page. The Controller returns the main page to the Interface with the specific input fields where the user can now input their configuration settings. Afterwards, the user selects the generate report button which starts the report generation process. The Interface passes along the user's inputs to the controller, which then calls upon the Engine to run the report generation algorithm using those user values. The Engine then returns this data to the Controller which, in turn, displays the information to the Interface as a report.

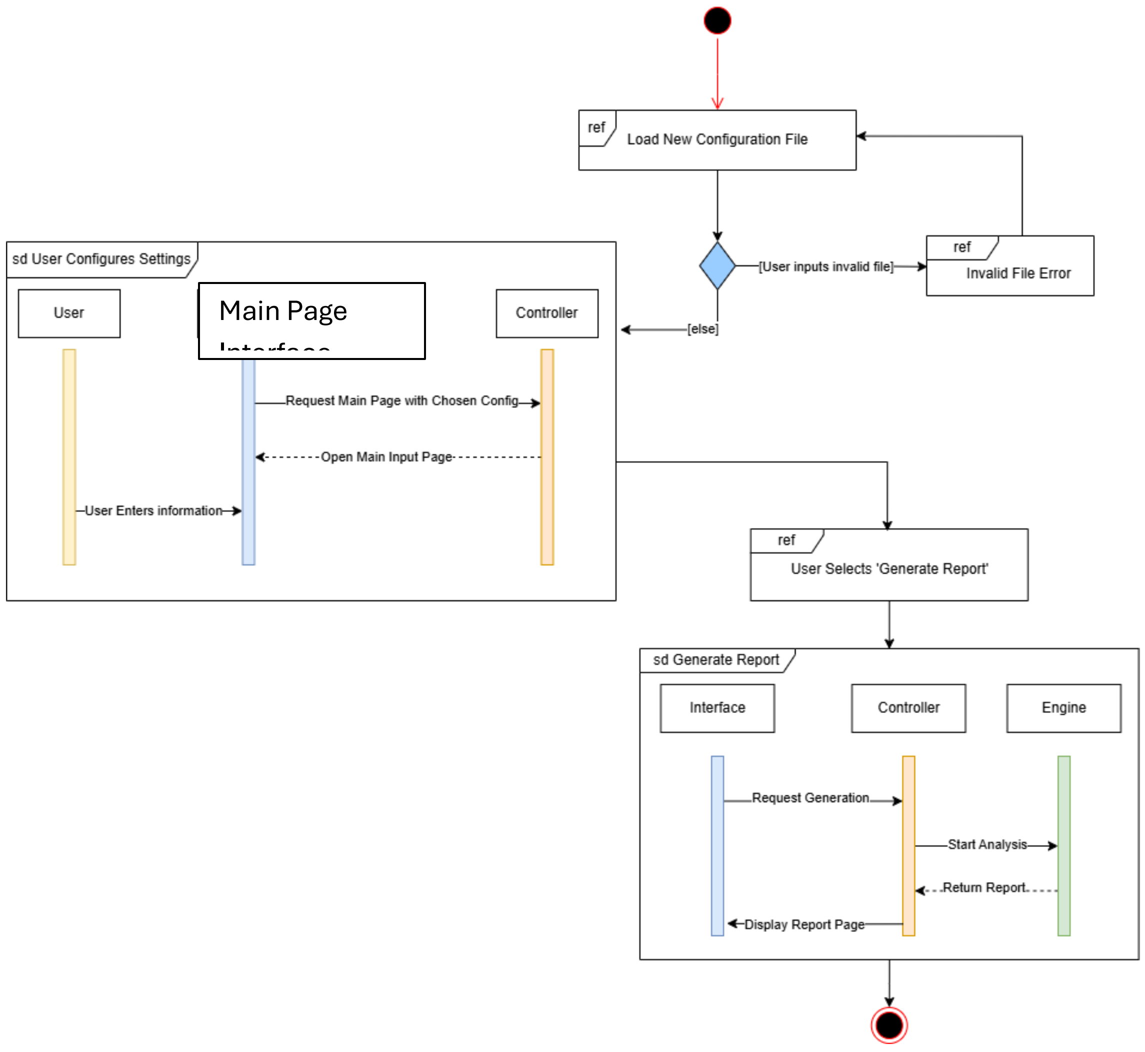


Figure 4: Dynamic Component Design Diagram

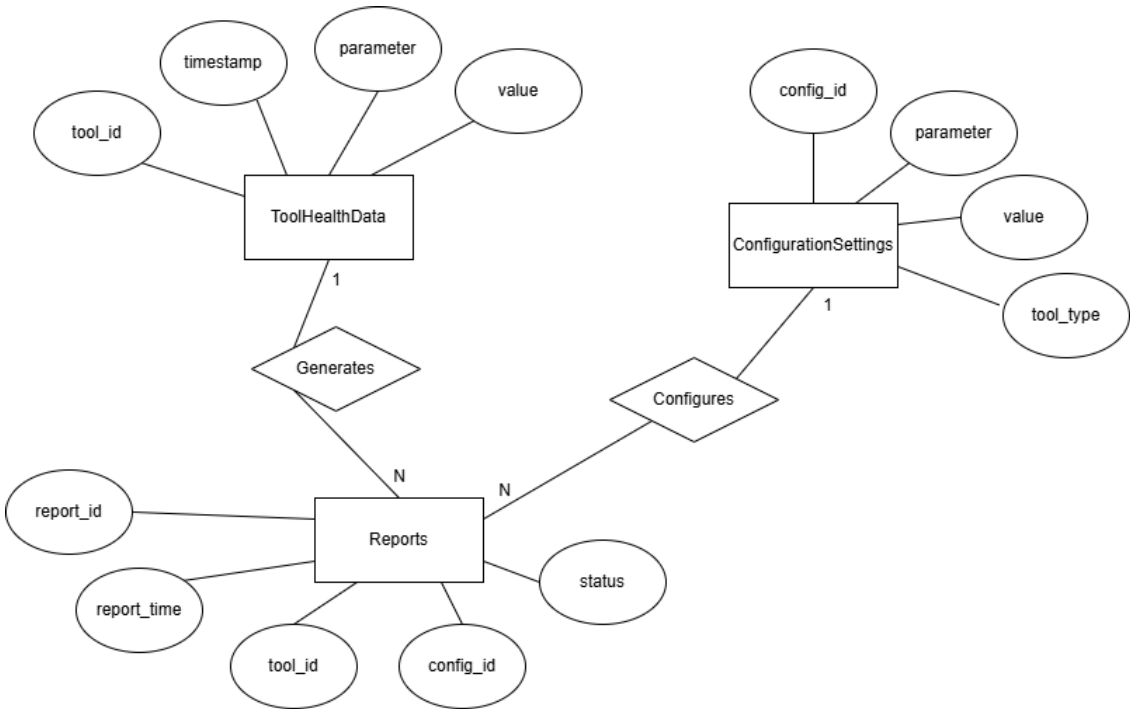
Data Design

Introduction

In this section, we'll explain how data is stored and exchanged in the WCM Tool Health Analyzer app, which uses SQLite as its database. SQLite is a lightweight, serverless database engine that's built directly into the app. It's fast and efficient for storing, retrieving, and managing data in tables, and it supports a variety of data types and relationships. All data will be stored locally on the user's device, so it's easy to access without relying on external servers.

Database Use

The WCM Tool Health Analyzer uses an SQLite database to handle all the important data. The database has several tables, each storing different types of information, like tool health data, configuration settings, and reports. Below is an ER Diagram that shows the main entities in the system and how they relate to each other.



Tool Health Data: This table stores information related to the health status of the tools used in the drilling process. It includes data like tool IDs, health status reports, and metrics collected from the tools during use.

Configuration Settings: This table holds the user-specific configuration settings used for data analysis, such as parameters for report generation, analysis thresholds, and tool preferences.

Reports: The reports table stores the generated health reports. It links to the relevant tool health data and configuration settings, and each report has a unique ID, timestamp, and status.

File Use

The application uses different file formats to handle input and output data. The most common file types include:

CSV (Comma-Separated Values): Used to import tool health data and configuration settings.

PDF: Used to generate and save reports.

SQLite Database File: Stores all structured data (like tool health data, configuration settings, reports, and user preferences) in a .sqlite3 file. This file is saved locally on the user's device.

Data Exchange

The application mainly exchanges data through files:

Input Files: CSV files are used to bring in tool health data and configuration settings.

Output Files: Reports are generated and exported as PDF files, which can be saved and shared.

Security

Security is very important, especially when dealing with sensitive data. Here are the main security measures in place:

Database Encryption: SQLite supports encryption for its database files, and this will be turned on to protect sensitive data.

Secure Data Transfer: All data transfers will use HTTPS to ensure that data is encrypted when sent over the internet.

Personally Identifiable Information (PII): If the system collects any personal data (like user details), it will be securely stored using encryption, and access will be restricted based on user roles.

UI Design

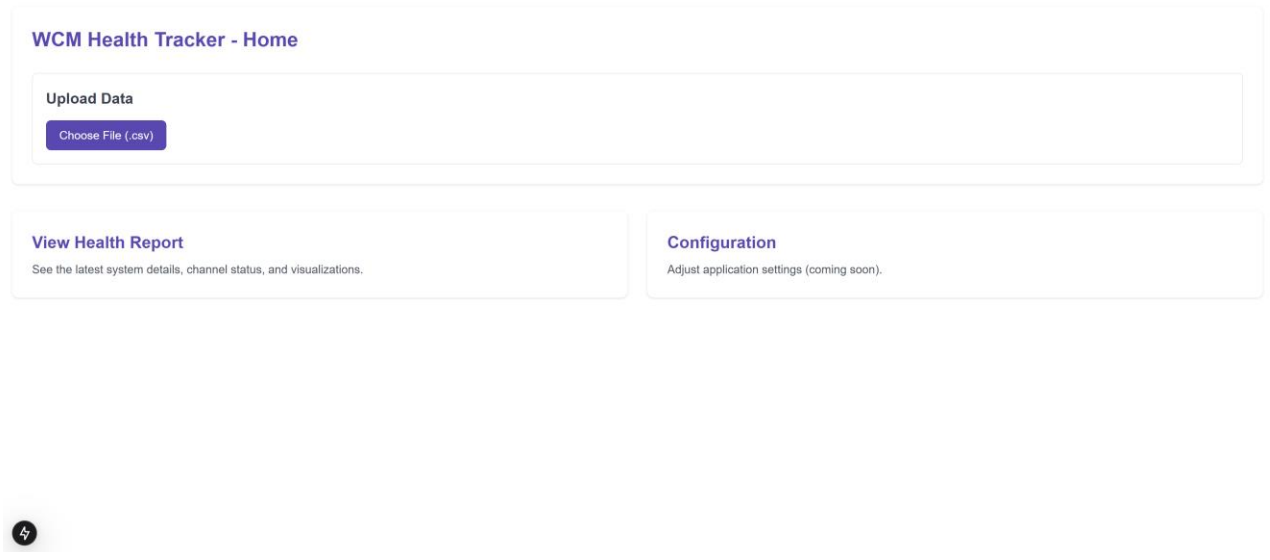
Introduction

The Tool Health Manager is a web-based application designed to help users monitor, manage, and analyze the health of tools used in well construction. The platform streamlines the process by allowing users to upload CSV files containing tool data, view and manage detailed health reports, and configure settings that influence report generation. The intuitive user interface ensures that tasks such as adding, deleting, and modifying data in reports, as well as exploring visual trends, are seamless and efficient.

This section provides a comprehensive walkthrough of the application's major screens, offering detailed explanations of their functionality, supported by relevant screenshots. Each screen is accompanied by a heuristic analysis that highlights how key usability principles have been applied to enhance user experience. The section also explores design considerations, focusing on widget selection, layout optimization, and error handling, ensuring that the interface remains intuitive, efficient, and aligned with best practices in UI/UX design.

Home Screen

Screenshot:



WCM Health Tracker - Home

Upload Data

Choose File (.csv)

View Health Report

See the latest system details, channel status, and visualizations.

Configuration

Adjust application settings (coming soon).

The home page serves as the central hub of the Tool Health Manager, providing quick and easy access to the application's core functionalities. It includes a Data Upload option that allows users to import CSV files directly from the home page, with the uploaded data automatically reflected in the Health Report Page, where users can analyze, manage, and visualize tool health data. Additionally, there is a link to the Configuration Page, where users can set parameters that affect report generation. The layout is designed to ensure that users can navigate to these essential features effortlessly, enhancing the overall efficiency of the application.

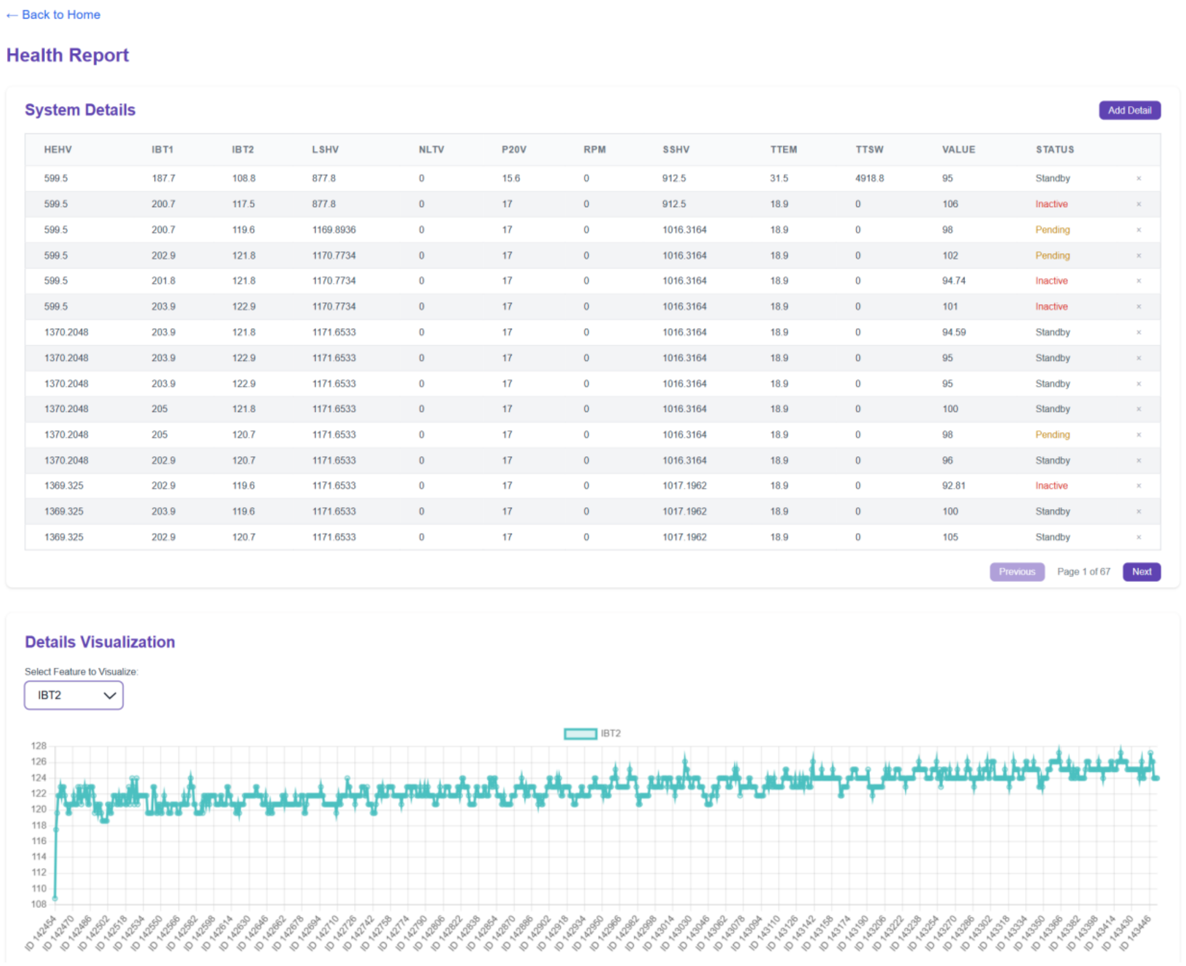
Heuristic Analysis – Home Screen

- **Visibility of System Status**
  - The system provides immediate feedback when a file is uploaded, displaying a confirmation message or error notification. This ensures users are always informed about the success or failure of their actions.
- **Match Between the System and the Real World**

    - The terminology and layout on the home page, such as “Upload Data,” “View Reports,” and “Configurations,” align with the user’s expectations and are simple and intuitive, making it easy for users to understand what actions they can take.
- **User Control and Freedom**
    - Users can easily navigate between the different sections of the app through clearly labeled links. They can also upload new data or revisit sections like the Health Report Page or Configuration Page without losing progress, ensuring flexibility in how users interact with the system.

Health Report Screen

Screenshot:



HEHV	IBT1	IBT2	LSHV	NLTV	P20V	RPM	SSHV	TTEM	TTSW	VALUE	STATUS	
599.5	187.7	108.8	877.8	0	15.6	0	912.5	31.5	4918.8	95	Standby	×
599.5	200.7	117.5	877.8	0	17	0	912.5	18.9	0	106	Inactive	×
599.5	200.7	119.6	1169.8936	0	17	0	1016.3164	18.9	0	98	Pending	×
599.5	202.9	121.8	1170.7734	0	17	0	1016.3164	18.9	0	102	Pending	×
599.5	201.8	121.8	1170.7734	0	17	0	1016.3164	18.9	0	94.74	Inactive	×
599.5	203.9	122.9	1170.7734	0	17	0	1016.3164	18.9	0	101	Inactive	×
1370.2048	203.9	121.8	1171.6533	0	17	0	1016.3164	18.9	0	94.59	Standby	×
1370.2048	203.9	122.9	1171.6533	0	17	0	1016.3164	18.9	0	95	Standby	×
1370.2048	203.9	122.9	1171.6533	0	17	0	1016.3164	18.9	0	95	Standby	×
1370.2048	205	121.8	1171.6533	0	17	0	1016.3164	18.9	0	100	Standby	×
1370.2048	205	120.7	1171.6533	0	17	0	1016.3164	18.9	0	98	Pending	×
1370.2048	202.9	120.7	1171.6533	0	17	0	1016.3164	18.9	0	96	Standby	×
1369.325	202.9	119.6	1171.6533	0	17	0	1017.1962	18.9	0	92.81	Inactive	×
1369.325	203.9	119.6	1171.6533	0	17	0	1017.1962	18.9	0	100	Standby	×
1369.325	202.9	120.7	1171.6533	0	17	0	1017.1962	18.9	0	105	Standby	×

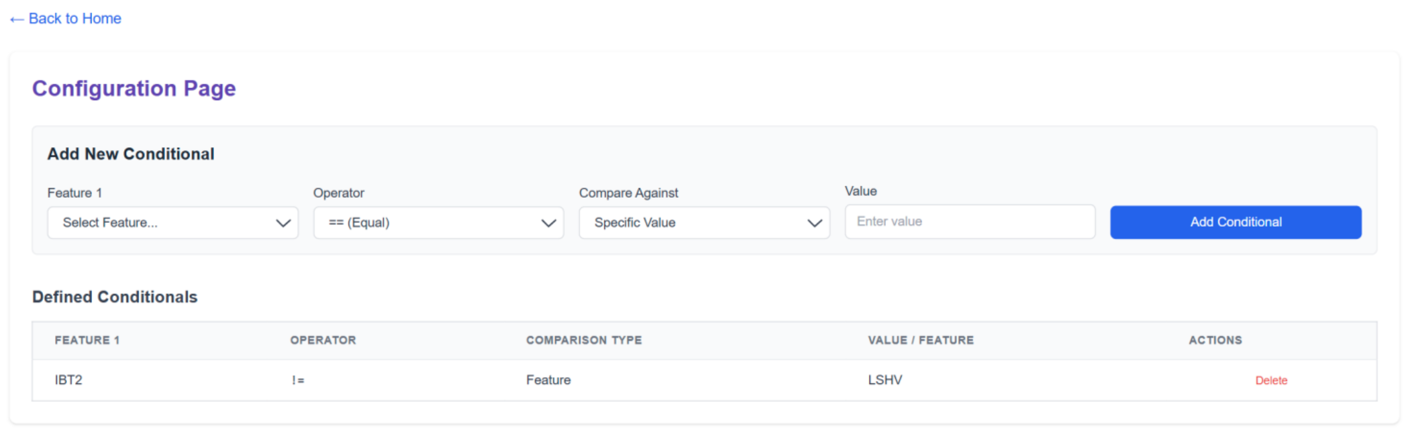
The Health Report Page allows users to analyze and manage tool health data generated from uploaded CSV files. It features a dynamic data table that displays relevant information, with options to add, edit, and delete individual entries. For large datasets, the table includes pagination to ensure smooth navigation and maintain performance. Additionally, a visualization section provides graphical insights, enabling users to easily identify patterns and trends in the tool data. The page is structured to allow efficient data management and visualization, helping users make informed decisions based on the uploaded data.

Heuristic Analysis – Health Report Screen

- **Aesthetic and Minimalist Design**
    - The Health Report Page is designed for readability and ease of use. The data table is organized with ample spacing between rows and columns, making it easy for users to quickly scan and interpret information. Pagination is included to keep the data manageable for users, preventing the table from becoming overwhelming. The color-coded statuses further highlight key information, improving visual clarity and allowing users to quickly assess tool health.
- **Recognition Rather Than Recall**
    - The Health Report Page minimizes the need for users to remember specific actions or data points. The option to delete an entry is always visible, with an "X" next to each row, making it easy for users to recognize that they can remove any entry at any time. Additionally, the add button is always available at the top of the page, ensuring that users can easily add new entries without having to search for the option. This consistent placement of actions helps users quickly recognize what they can do without having to recall or navigate through additional menus.
- **User Control and Freedom**
    - Users have full control over the data within the table. The ability to add, delete, and navigate through entries ensures that users can easily manage and adjust their data as needed. The straightforward interface allows users to make changes without unnecessary restrictions, giving them the freedom to tailor the report as they see fit.

Configuration Screen

Screenshot:



FEATURE 1	OPERATOR	COMPARISON TYPE	VALUE / FEATURE	ACTIONS
IBT2	!=	Feature	LSHV	Delete

The Configuration Page provides users with input fields that allow them to define parameters that influence the generation of health reports. Users can specify threshold values, adjust filtering criteria, and customize settings to tailor the report output to their specific needs. The page is designed for flexibility, enabling users to apply changes easily or revert to default settings as needed, ensuring that reports are generated according to the desired criteria.

Heuristic Analysis – Configuration Screen

- **Visibility of System Status**
    - When users make changes or save settings, the screen gives clear feedback like messages for successful saves or errors. This helps users know what's going on right away.
- **User Control and Freedom**
    - Users can change settings anytime and can also reset everything back to default if needed. This makes the screen flexible and user-friendly.
- **Recognition Rather Than Recall**
    - All options are easy to see and labeled clearly. Users don't have to remember where things are or guess what to do. They can easily recognize what they need right on the screen.

Appendix

Michael Cao

Contact Info: mcao77@gatech.edu

Contributions: Applying Data Filters to cover edge cases

Arun Murali

Contact Info: arumnurali@gatech.edu

Contributions: File Export Functionality

Justin Kamina

Contact Info: jkamina3@gatech.edu

Contributions: -Adding Pass/Fail Functionality

Chenming Fan

Contact Info: cfan70@gatech.edu

Contributions: Visualization generations

Manuel Morin

Contact Info: mmorin8@gatech.edu

Contributions:

- Data Input & Processing
- Early Algorithm Development
- Pagination support