Golden Fire Department

Volunteer Firefighter Information Tracking

June 18th, 2019

Hannah Levy

Bradley Helliwell

Heidi Hufford

Connor Brennan

Ty Christensen

Contents

1 Introduction

The Golden Fire Department is a volunteer department, serving the 50,000 people who live, work, and learn in Golden. The fire department is responsible for rescues, emergency medical services, fires, service calls, and hazardous conditions across the nine square miles of Golden. Comprised of 87 volunteer firefighters and 12 administrative staff, the department responds to more than 100 calls each month, which totaled to more than 2,000 calls in 2018.

The Golden Fire Department faces severe staffing issues, suffering from a lack of both coverage and consistency. The coverage issues result from the high proportion of volunteer emergency response staff, causing the Golden Fire Department to have particularly low staffing during business hours when the volunteers are at their regular jobs. Staffing inconsistencies result from procrastination, making the fire department shorthanded until the end of the year when volunteers are rushing to meet their annual requirements. For the fire department, chronic understaffing causes an increased response time, especially during overlapping calls, when there are often too few firefighters to fully staff each response.

In addition to problems with staffing, the fire department uses inefficient means to track progress for each of their firefighters. To remain in good standing, volunteers must meet a set of requirements by the end of the year. Their progress reports involve days of manual data entry from multiple sources and are often inaccurate due to human error. Because the reports are so time-consuming to construct, administrators and firefighters are only able to review their progress a few times a year, making it impossible to address individual staffing issues on a regular basis.

To solve this problem, the Golden Fire Department requested a system to generate these reports for the administrators and officers and to create easily accessible progress reports for individual firefighters. The team's solution was to create a web-based application and database to pull, process, and display the necessary information with minimal administrative labor.

2 Requirements

Functionally, the Golden Fire Department Information Tracker is required to facilitate data access and generate accurate progress reports for each of the fire department's staff members. Below, each of the specific requirements is listed as determined in weekly meetings with the Golden Fire Department.

2.1 Functional Requirements

- **Website**–All information should be accessible online.
- **Design**–The website must have an appealing color scheme with few white backgrounds to prevent eye strain.
- **Dashboard**–The website should display a dashboard interface for administrators and officers.
- **Documentation**–Each view of the website must contain an about page to explain how to use the website.
- **Reports**–The system must generate accurate reports regarding progress on requirements and length of service.
- **Firefighter Statuses**–Firefighter progress must be measured with statuses of either Complete, On Track, Falling Behind, or Behind.
- **Security**–The website must be secure and contain a login system.
- **Data Access**–Administrators should be able to view and edit data for all firefighters, officers should be able to view information for firefighters, but not edit it, and firefighters must only be allowed to view their own data.
- **Data Storage**–The database should be able to handle up to seven years of prior data as well as all necessary future data.
- **Data Entry**–Administrators must be able to initiate a database refresh, upload IAmResponding reports, and manually enter or remove status changes for firefighters.
- **Automated Data Entry**–The database should automatically update data from Emergency Reporting upon administrator login.

2.2 Non-Functional Requirements

- AWS–The website must be hosted on Amazon Web Services.
- Emergency Reporting–The system must collect data from Emergency Reporting through its API.
- Security–The system must implement basic database security procedures.
- Modular Design–The project should be written using a modular approach to allow for upkeep and extension.

3 Project Overview

The Golden Fire Department Information Tracker pulls information from two of the department's data services, Emergency Reporting, and IAmResponding. Depending on the staff member's title, they are presented the data in one of three views, as detailed in the following sections.

3.1 Administrator View

Administrator access is granted to the fire chief, the executive assistant, and other managerial staff. Administrators can access six pages: Home, Reports, Custom Reports, Personnel, Import, and About. As shown in figure 1, the homepage shows a dashboard with a snapshot of firefighter progress and year-to-date statistics about the call volume and response availability. The report page allows administrators to generate either standard or LOSAP (Length of Service Award Program) reports for either a single firefighter, a group of firefighters, or the entire department. On the custom reports page, administrators can view time logs for any employee. The personnel page allows administrators to log status changes, and the import page can be used to import reports from IamResponding and initiate a database refresh for data stored in Emergency Reporting.

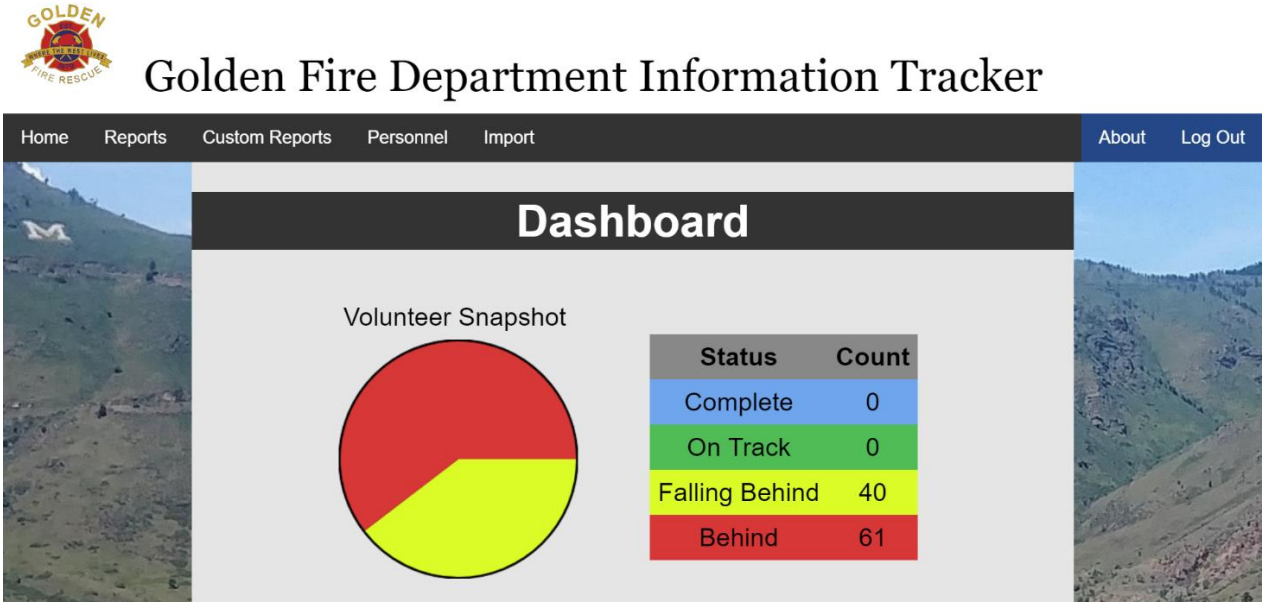

Figure 1: Administrator Homepage

3.2 Officer View

The officer view is designed for supervisory staff members who oversee volunteers. The pages that they can access include the Home, Reports, and Custom Reports, which each have the same functionality as the administrator view.

3.3 Firefighter View

The firefighter view allows volunteers to view their own progress and was designed to motivate them to volunteer consistently throughout the year. As shown in figure 2, the top of their homepage shows the firefighter's status. This status is calculated based on the completion status of each of their six requirements: training, work detail, apparatus checks, business meetings, fundraisers, and either calls or shifts depending on their residency status. Below the general status is a progress chart, showing completed requirements in blue, incomplete expected progress in red, and gray indicates requirements that are not expected to be completed. The table below the general status, as shown in figure 3, indicates a volunteer's progress towards each of their requirements, each with charts showing where they stand.

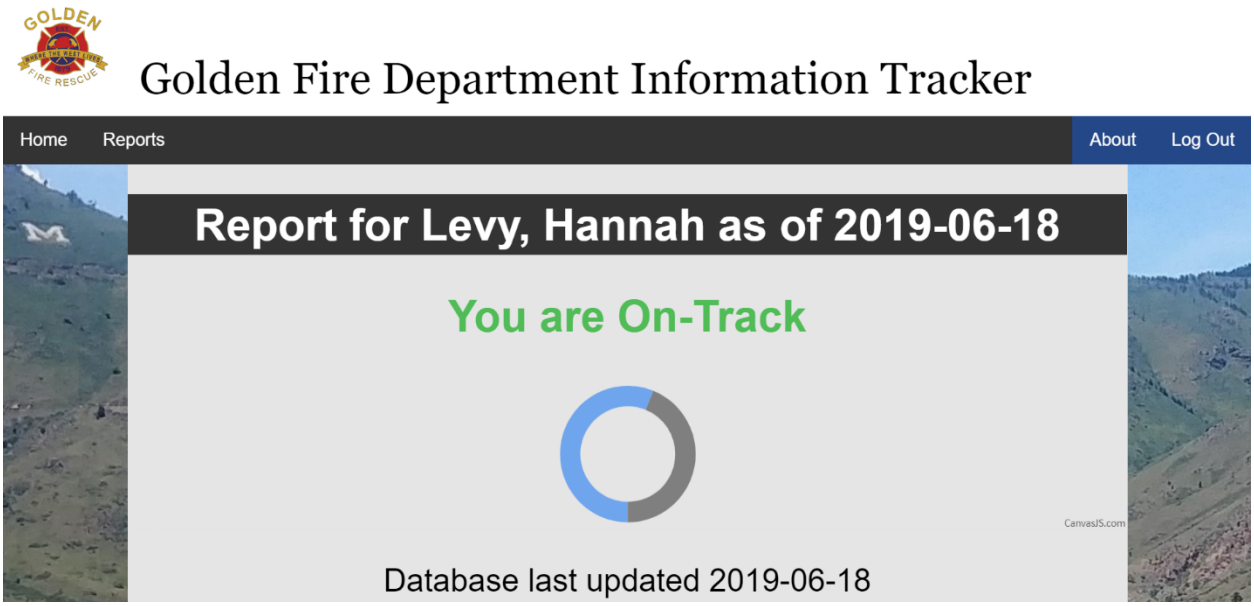

Figure 2: Firefighter homepage

Figure 3: Firefighter progress chart

4 System Architecture

The system architecture is divided into three main sections: a Django framework, an HTML organization, and a database. The following sections detail the design of each section.

4.1 Django Framework

At the core of the system lies the Django framework, which is responsible for integrating the HTML, CSS, and Python files. At a high level, the Django project is divided into a central project module and several apps, each corresponding to one core function of the website. Each app contains a views.py and urls.py file, along with HTML templates. The views Python file determines what behavior should occur if a user navigates to or interacts with a webpage in the app by handling the HttpRequest and HttpResponse pairs. The urls.py file is used to send the HttpRequest objects to the appropriate views.py function based on the URL being accessed. The urls.py mappings also capture pieces of the URL and use them as parameters in the views functions. The views.py function takes in the information from the call from urls.py and builds an HttpResponse object, which it then returns. In this project, HttpResponse objects are built in one of two ways. In the first and most common way, the views.py function loads in an HTML page to be used as a template. It then processes data from the request and function parameters as needed and creates a dictionary of context containing name-variable mappings for data that needs to be passed into the HTML. The function then returns an HttpResponse object with the rendering of the template with the context and the initial HttpRequest object. In the second way, the function creates an HttpResponse object with the content_type set to 'text/csv'. It then builds a CSV file in the HttpResponse. When this HttpResponse is returned, the result is the CSV file being downloaded onto the user's computer. These two methods reflect how the system handles rendering webpages and generating CSV reports that can be downloaded by the application user.

The central project module contains the information necessary for the project to run. The module contains the settings.py, which file contains information about allowed hosts, acceptable middleware, logging, and the location of the WSGI application. Additionally, each app in the project is registered in this file. The module also contains the manage.py file which is used to launch the application locally, create new apps, and collect the static files, amongst other core functionality. The central project module also has its own urls.py file, which contains URL mappings to each of the apps and their urls files.

Outside of the apps and the central project module, there are three other directories that contain key components of the application. The templates directory contains the HTML templates, and the static directory contains CSS and image files used by the website. The source directory contains all functions and classes used to access the database and the Emergency Reporting API and process the data.

4.2 HTML Framework

The HTML files define the basic functionality of each webpage. As shown in figure 4, there are five main pages: Home, Reports, Custom Reports, Personnel, and Import. The common functionality for each view (administrator, officer, firefighter) is stored in a base file. The HTML files for each individual page inherit functionality from these base files. This code hub helps lower the amount of repeated or unnecessary code and helps allow for modular development. The CSS files are also all kept in one folder. Each page on the website grabs from these CSS files to determine their font size, text color, padding, and other important styling aspects. Keeping the CSS and HTML in separate files helps the code to be modular and consistent.

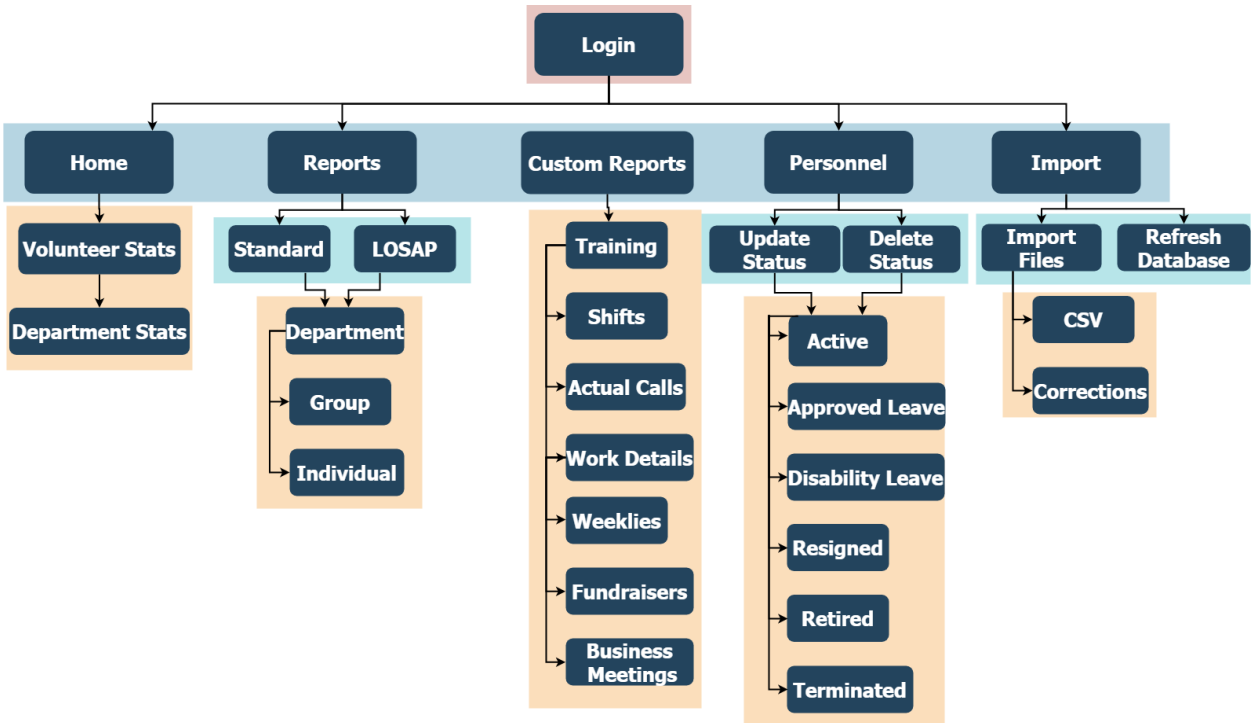

Figure 4: Website Flow Diagram

Since administrators, officers, and firefighters each have a different view of the website, there are different sets of pages based on their access levels. Some pages are specific to administrators, like the personnel and import pages. Because of this, each navigation bar has different tabs. Since error pages also have a navigation bar, this required the team to create three different error pages based on the three access levels. Although there are different views, each page grabs from the same image folder, containing the background images and the fire department's logo. In general, the HTML framework was constructed to reduce repetition and simplify the process of developing consistent style and functionality.

4.3 Database Design

The database is designed to hold all the information needed to generate reports and support the functionality of the website, using five main entities, as shown in figure 5.

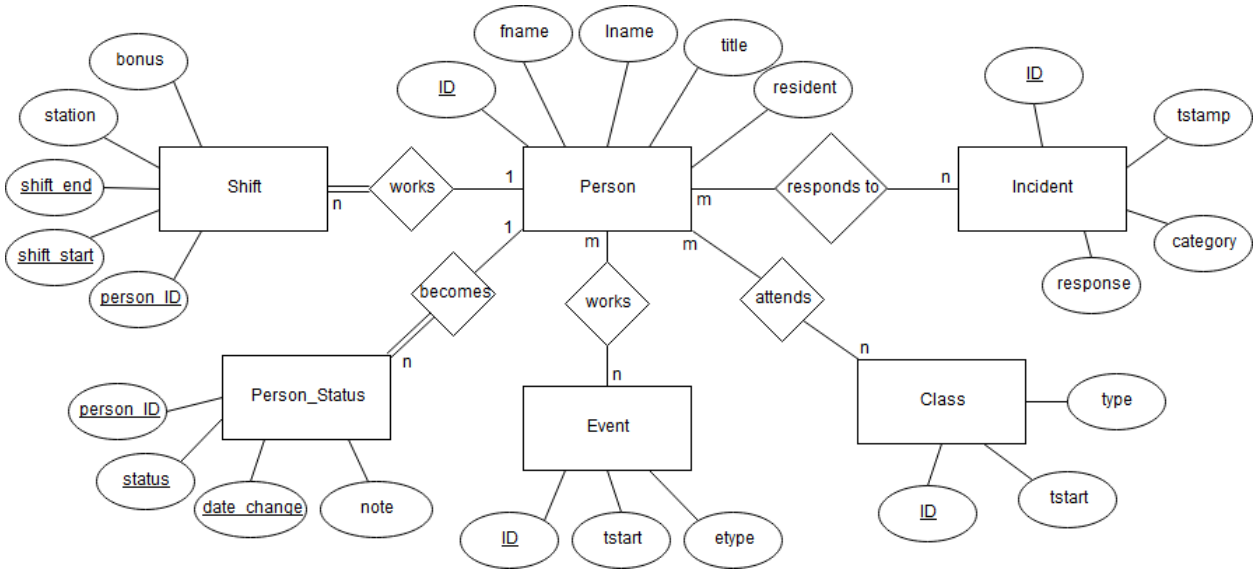

Figure 5: Database ERD

The Person entity is a sort of hub. Every other entity is related to this one in some form. It tracks a person's official employee ID, their name, their current rank in the department (such as firefighter, shift officer, or chief), and whether they are considered a resident.

The Shift entity tracks all shifts worked by firefighters. Due to the loose scheduling structure this department employees, each shift object is specific to one and only one person. It tracks that person, the start and end times of the shift, which station the shift was worked at, and if there was any associated bonus (such as a shift being worked on a weekend or holiday).

The Person_Status entity tracks each firefighter's status changes. When a person is first entered into the database, they will typically be given an "active" status change, which marks the beginning of their service. New entries will be added any time a person's status changes (such as going on leave, returning from leave, or retiring), forming an easily accessible service log. It tracks the person a status change is associated with, what the status is, the date it was changed, and has an optional note field if the administrators want to keep track of additional information regarding the change.

The Event entity stores information regarding work details, fundraisers, and business meetings. It tracks an ID generated by emergency reporting, the start time for the event, and the event type. There is a cross-reference table that connects event and person so that each person can attend many events and vice versa.

The Class entity is used to track training events. Since these events were stored in a different place in emergency reporting, their ids could overlap with ids from events, so they had to be made a separate table. It tracks an ID generated by emergency reporting, the start time for the class, and the class type. There is a cross-reference table that connects class and person so that each person can attend many classes and vice versa.

The Incidents entity holds information regarding incidents that the fire department has responded to. It tracks an ID generated by emergency reporting, a timestamp for when the incident started, an incident type, and the response time. There is a cross reference table connecting it to the person, because each person responds to many incidents, and incidents are responded to by multiple people.

5 Technical Design

5.1 Data Handling

The majority of the data used by the team's web application originates from Emergency Reporting, which stores information regarding personnel, training, incidents, and work details. Before being stored in the database, Emergency Reporting's information is collected using their API. Initially, the API caused long loading times each time the database updated. To resolve this, the API is called when pulling new information and is compared against the information already in the database. If the two sets contain the same data, nothing is done. Otherwise, the new information is uploaded. This reduced the amount of time the system spends waiting on network communication by putting more load on the server's hardware. This proved to be much more efficient than making multiple API calls.

Unlike the other entities, the data in the Shifts table can only be obtained from IAmResponding. Since IamResponding does not have an API, the only way to retrieve the data is to manually generate a scheduling report and then upload it to the Golden Fire Department Information Tracker. The contents of IamResponding's reports are processed into a data frame using the panda's Python module. From the data frame, the Python code parses and reformats the information before feeding it to the loader function. As a result, all the users need to do is run a report and upload it to the web app's import page.

5.2 User Authentication

To protect the Golden Fire Department's data, a system for user authentication was needed. The easiest and most effective way for the team to integrate this was to pull from Emergency Reporting's user authentication system. The base URL for the project was directed to a login screen. The clients are able to sign in with their Emergency Reporting credentials. Once this data is entered, the system prompts the Emergency Reporting API for an access token and a refresh token based on these credentials. If the API returns a successful response, the user is considered authenticated and can continue

onto the website. The access token is then used to retrieve the user's title and employee number from the API, which is then used to redirect the user to the correct page. The refresh token is sent in the URL whenever a request is made to move to a different page. Upon entry to the page, the refresh token is pulled and sent to the API to verify that the user can still be authenticated. If the authentication is successful, the requested view is rendered. If the authentication fails, the user is logged out and prompted to sign back in.

6 Quality Assurance

Below is a list of quality assurance steps the team took to make sure the product not only met the functional and non-functional requirements but also provided accurate and up-to-date data.

Software Quality Plan:

- Unit testing Python code with, including functions that interact with the database
    - Using the Python Unittest library
- Website Tests
    - Ensuring that all links direct to the correct pages
    - Testing the forms on each page to verify that they function properly
    - Ensuring that unexpected inputs do not cause unintended problems
        - Inputs are validated before being sent to the database, so they are guaranteed to be quality
    - Opening the webpage in different browsers (Edge, Chrome, Firefox, Safari) to verify that the website's style and functionality remains consistent.
    - Ensuring that all CSVs download correctly
    - Validator for HTML, CSS, and JS (W3C validation service, shown in figure 6)
    - Testing API output against expected results

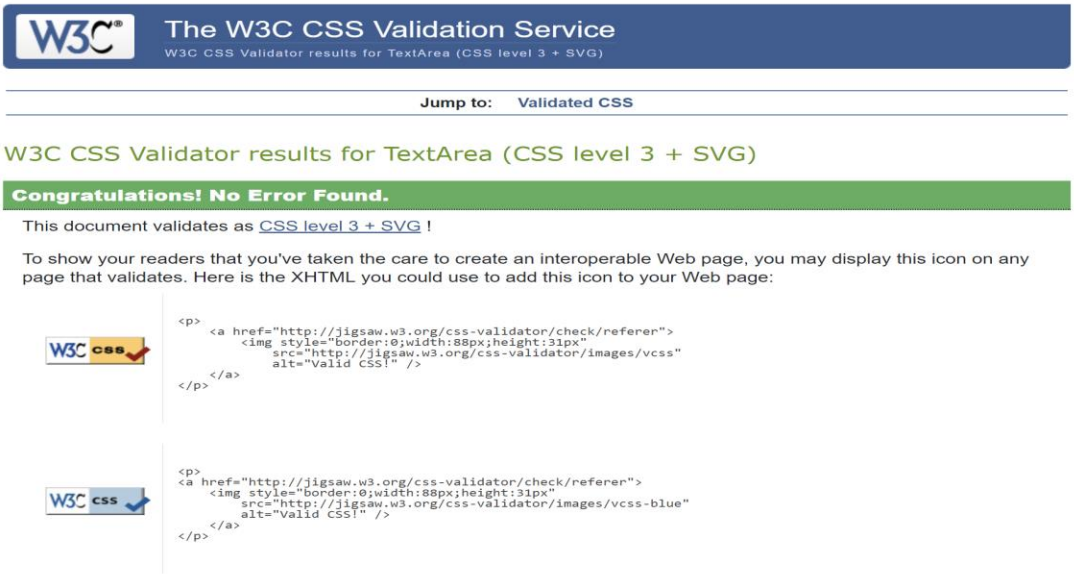

Figure 6: CSS Validation

- Following good code standards
    - Maintaining only one CSS id per HTML page
    - Lower case element and attribute names
    - Consistent spacing and indentations within code
    - Comments where necessary
- Demo to clients, get feedback, and ensure the product is on-track (user-acceptance testing)
    - Easy navigation around the webpage
    - General appearance, no white backgrounds, larger fonts, readable color scheme
    - Making sure the web page fits different screen sizes and screen resizes
        - Specifying size by % and vw instead of pixel width/height
    - Mobile compatibility
- Pair programming and team development ensures good software development

7 Results

7.1 Unimplemented Features and Future Work

One desired feature left unimplemented was total automation for data collection. Ideally, the system would have used APIs to access all data and those APIs would be used to update the database every night at midnight through a function hosted in AWS Lambda. This feature was not implemented due to resource limitations and security concerns. One of the websites the system needs to access, IAmResponding, does not have an API, so an administrator will need to download a report from that site and upload it to the tool. Emergency Reporting does have an API, and the fire department would have liked the team to automate updating the database on a nightly basis, but this would require hard-coding an Emergency Reporting username and password in the code, which introduces security risks and additional complexity. As a compromise, the database updates when administrators log in and older records can be added to the database using a button on one of the pages.

For future work on this project, the Golden Fire Department Information Tracker could be extended to construct reports containing descriptive statistics about the fire department. Initially, the fire chief had discussed with the team the possibility of generating such reports, which would be persuasive tools to help the department receive funding to hire additional career firefighters. Extending the project to include this functionality would involve accessing reports from When2Work and writing queries to generate these reports.

7.2 Testing Results

The web application parses through a large amount of data that is generated by the Emergency Reporting API. Originally the code relied heavily on many requests for information from the API which proved to be incredibly slow. Since this code is being run when an administrator logs into the web application, this was unacceptable. To resolve this, an algorithm was implemented to reduce the number of API calls and rely more on computation. One example of this is used in the function that populates the database

with incidents. Previously, an API call was made to retrieve each incident, but performance testing showed that this took too long and was not feasible. Now the API is called four times to gather all the information needed and then search algorithms are applied to a list to find the desired information. Performance testing showed a significant improvement with a reduced load time from more than six minutes to less than ten seconds.

To test the user interface, the team tested the product website in multiple different browsers, including Chrome, Firefox, and Edge to determine if the formatting was consistent and easy to read. Although the tool was not written as an app, the team tested the site on mobile devices to ensure that firefighters could access it from their phones or tablets.

When performing usability tests, the team received a positive response from the firefighters and administrators at the fire station. The administrative assistant was excited to see how quickly the tool could generate reports for all staff members. Additionally, each administrator who tested the website confirmed that the tool functioned in the way they needed it to, and the chief commented on how much she liked the graphics. When a firefighter logged into the website, he commented that he thought it was simple to use and easy to read. After giving them the link, they distributed it to the rest of their staff members, who identified several bugs and provided feedback about the color-coding on the firefighter homepage. Listening to user input made the website more user friendly and have fewer errors.

7.3 Lessons Learned

Completing this project made it clear how important it is to research and select services wisely. One of the City of Golden IT Department's employees suggested the team use Amazon Web Services (AWS) to host the website. Initially, the team trusted his suggestion and attempted to host the website on AWS but found that the documentation was outdated and that receiving assistance from Amazon was difficult. The team learned that more extensive research should have been done; no one at the

Golden IT Department had ever used AWS and had just recommended it based on Amazon's reputation. Had alternative services been researched, the team could have saved several days' worth of work.

Another lesson from this project is that maintaining product and sprint backlogs are important. While working on the project, the team found that it was easy to prioritize, plan, and allocate tasks when there were backlogs. Having a shared and documented list of remaining tasks also made the team more productive because it was easy to see how the team was progressing and how much work was left.

The last lesson learned from this project was the importance of communication. Specifically, when performing usability tests, it's important to communicate the functionality that the users can expect to see, and what functionality has yet to be added. When the team gave the fire department the URL to the website, they distributed it to their employees and asked for feedback. The team received several reports that parts of the website were not working for parts that the team hadn't completed working on yet. The team learned to be very clear about what the user should expect so that their feedback is more relevant.