Municipal Solutions

Joseph Hines
Computer Science
Drexel University
Philadelphia, Pennsylvania, USA

1 Abstract

Public Works Crews and highway departments around the world are frequently tasked with large scale road closures. The most common reasons are due to local events such as parades and car shows that use the streets to host their events. Municipal Solutions aims to provide a platform that enables foremen and superintendent's to efficiently plan these road closures. Users are able to log into the system and create projects. When working on a project, users can interact with a map of the necessary roads to place their resources. Along with this, they can create and assign resources to personnel, ensuring that every barricade and truck has someone that is responsible for it. Finally, they can create a plan for the shut-down and be confident that someone will take care of each resource along the way. All of this data can then be exported and shared with all of the relevant parties.

2 Introduction

Planning public events that involve road closures and cross-department coordination is a difficult task. Identifying the proper place for road barricades, vehicles, and personnel is a non-trivial task. Once a solution is found, it is crucial that it be communicated well, especially when other departments are involved. When it comes time to close down the streets and execute the plan, there is no place for confusion or miscommunication, as the potential impacts on traffic can be unsafe. Municipal Solutions strives to provide a platform that allows public works foreman to excel in this process, from start to finish.

Users can create projects and interact with a map of their location for planning. They can layout road barricades and personnel on the map, which allows them to quickly create and visualize their plans. The interface keeps track of all the resources and personnel used and reports these totals back to the user so that they can ensure that they have all the necessary resources before moving forward. A user can then create their shut-down/open-up plan by assigning certain resources and locations to different personnel. All of this information can then be printed out in an auto-generated document and sent to the involved parties. This takes the majority of the hassle and work out of the planning procedure, and minimizes communication between departments when it is time to execute the plan.

3 Background

It is critical for this system to have an intuitive user interface that also is powerful enough to meet the demands of users. There has been a lot of work done with regards to enabling a first time user to accomplish their task while also empowering the power user to take control of their workflow (Lafreniere et al. 2014, Stuerzlinger et al. 2006, Bruno et al. 2005, and Neto et al. 2009). Given the web platform and the variety of possible screen sizes that users may have, this work is a valuable reference for getting the most out of the screen. Lobo et al. 2015, Lanning et al. 2014, Dunnavant. 2010, and Church et al. 2010 all provide interesting insights into digital maps, how to display them, and how people interact with them. This will provide insight into how we can best approach the design of the map interface. There has also been a lot of interesting work done with regards to digitally analyzing traffic and road closures, which can be used when reviewing potential issues with proposed plans (Pietrobon et al. 2019 and Wang et al. 2019). Pollack. 1995, Denoue et al. 2014, and Hassani et al. 2018 all have done work related to digital planning, project management and communication. The lessons learned here will directly contribute to our overall approach at creating our platform.

4 System

This is a relatively simple system. In order to provide the desired service to the user, a few pieces are needed. Flask is being used as the webserver, serving the pages and communicating with the database. The database of choice is

sqlite3. The pages served to the user are filled in using Flask's built in jinja template system. Most pages are just HTML, but a few include JavaScript. These pages utilize jQuery for DOM manipulation and the OpenLayers API for map rendering.

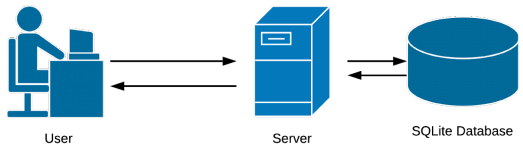

Figure 1: High-level overview of server architecture

4.1 Server-side

Using python3, the server leverages Flask for handling web requests. The server serves the different pages, and provides a single API endpoint. Users can access a few pages without having to create and sign into an account. Such pages are the home, about, login, and sign up. These pages are simple HTML documents that display some information and offer links to other pages. The home page contains some simple copy and a flashy graphic along with a large button prompting users to sign up. The about page contains a short explanation of the application and the motivation behind it. The goal of both of these is to funnel users towards the sign up page

From the /signup page, a user can provide an email address and password. The server verifies that the email address is not already in use before encrypting the password and storing the user in the database. This is done by leveraging a simple Database object that provides an interface for interacting with the underlying sqlite3 engine. Once the user is successfully saved, they are sent to the /login page, where they enter their credentials again. The server verifies that the provided password encrypts to what the original password encrypted too, and then logs the user in. The user is then sent to the /dashboard page, where the user can see all of the projects they have created as well as create more. Clicking the "create new" button brings users to a new page, where they can provide a name and description for their project. From here, they are sent to the project page.

The core of the functionality is built into this project page. The page's location takes the form "/project/<PID>" where PID is the project id. The server parses this out, checks to see if a user is logged in, verifies that the user is the owner of the project, and then sends the populated project template back. From here, users can do all of the pertinent work that will be described in the following section. With regards to the server, there is on API endpoint that this page hits. This is the /api/saveproject page. The client hits the server with a POST request whose body contains the project id and the project state. The server takes this information, again verifies that the user making the call is the owner, and then updates the project information in the database. The final page is the /report/<PID> page, which provides a report about the relevant project. This page is designed to be printed and handed out to the crew involved in the operation.

The database involved in this application is fairly simple. There are two tables, users and projects. The user table stores email addresses, unique user ids, and the encrypted passwords. The project table stores the names, ids, descriptions, and data. The data field contains the state JSON of the project stored as a string. The Database object ensures that these tables are created when it is initialized. It provides methods for inserting into the tables, updating their records, and retrieving records from them.

4.2 Client-side

The client-side of this application, much like the server, is largely simple. The only portion that needs discussion is the /project page. The other pages are simple HTML templates that are populated with data by the server. The /project page is where the main application is implemented. **Figure 2** shows what this page looks like. At the core of the application is the global *state* object. This object is what is stored in the data field of the projects database. This object contains all of the information about the current state of the project, and is updated by nearly all user actions. Some data points are the selected tool, location, features already placed on the map, personnel added, and steps created. This object is treated as immutable, and is never mutated directly, only through the **setState** function. This helps prevent most mutability issues and allows the data to be incrementally updated in a way that ensures the currently displayed interface exactly matches its data representation. This function also contains a call to the **save** function, which makes the call to /api/saveproject. By sending a POST request with a body containing the project ID and current state object, the server is able to save the updated project information in the database.

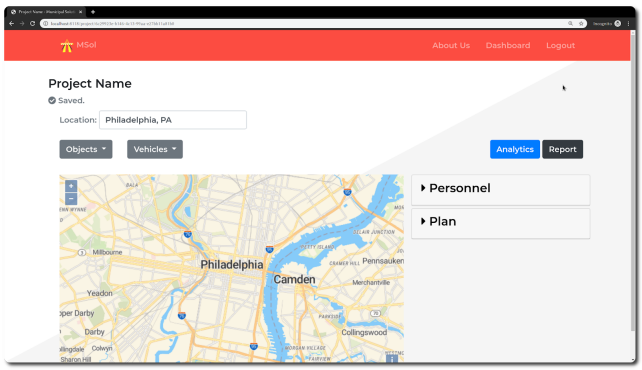

Figure 2: /project page

The **save** function is also wrapped in a **debounce** function, which is used to buffer subsequent calls to a function and only execute the last one. This allows a user to make many

subsequent actions, and only triggers a call to the database after a defined idle period. This drastically limits the amount of calls to the server, which leads to a smooth user experience and enables the server to scale better. Now that these functions have been covered, we can now look at what adds functionality to the application.

In order for this application to work, the map needs to have a certain amount of functionality. The map needs to be able to display roads, change locations, handle the placing of markers or features, displaying popups, and basic panning and zooming. Most, if not all, map libraries have this functionality. Each of the functions of the map have functions that adapt the application to API. Such functions include **addFeatureToMap**, **displayPopup**, and **bindMap**. The **bindMap** function is called shortly after page load, and handles binding map events to functions. For example, on click, the event coordinates are grabbed and passed to **addFeatureToMap**, which looks at the state and adds the current tool to the click coordinate, and adds the placed resource into the state object so that it can be tracked and assigned later. This function also handles displaying popups when a marker is clicked. This popup allows the user to change the name of the marker, see where that resource is assigned, and delete the marker. This is the main functionality of the map, which then enables the other elements of the interface.

Above the map, there is a search bar that utilizes geocoding to recenter the map. Just below this, there are buttons for dropdowns so that users can select different resources to place on the map. On the right side of the page there are two tabs, Personnel and Plan. These two tabs expand to reveal their relevant lists. Users may add to either list by clicking the add button that is always at the bottom of the corresponding list. These lists function largely the same. They allow items to be renamed, have resources assigned to them, and be deleted. The analytics tab opens a modal that displays lists of resources that are unassigned as well as an overall count of all the resources used. This all comes together in the Report button, which displays a report generated from the currently devised plan. This report contains a picture of the map, the analytics information, and the entire plan laid out, showing which people are responsible for what resources. This report is designed to be printed out so that it can be distributed physically or as a PDF.

5 Discussion

For mapping software, I decided to use the OpenLayers API. This API is well documented online, and provides all of the functionality necessary. For the data backend, I chose MapTiler. This service is free, has a clean style, and provides geocoding services as well. I found that the performance of the map to be sufficient for this application. I chose to utilize the OpenStreetMaps Geocoding API for location lookups. This was not due to any deficiencies in the MapTiler API, instead it was due to the OpenStreetMaps API needing much less parsing for my simple use case.

As for further improvements, I would have liked to revamp the assignment functionality. I think this could be improved by adding support for drag-and-drop, as well as adding a form of highlighting for markers so that users can clearly see what markers they are selecting. I think that this would better enable new users to learn the application as well as enable power users to get more done in less time. Aside from this, I think the overall interface may be reorganized to better communicate functionality to users. I also think that the report could use more work, so that the plan is communicated in the best possible way.

Overall, I have designed a system that I am proud of and I feel enables users to accomplish the desired task. As mentioned above, I am far from set on this being the best solution, but I feel that it is a step in the right direction as there is large need for software in this market.

REFERENCES

[1] Benjamin Lafreniere, Andrea Bunt, and Michael Terry. 2014. Task-centric interfaces for feature-rich software. In Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design (OzCHI '14). Association for Computing Machinery, New York, NY, USA, 49–58. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/2686612.2686620

[2] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. 2006. User interface façades: towards fully adaptable user interfaces. In Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06). Association for Computing Machinery, New York, NY, USA, 309–318. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/1166253.1166301

[3] A. Bruno, F. Paternò, and C. Santoro. 2005. Supporting interactive workflow systems through graphical web interfaces and interactive simulators. In Proceedings of the 4th international workshop on Task models and diagrams (TAMODIA '05). Association for Computing Machinery, New York, NY, USA, 63–70. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/1122935.1122948

[4] Americo Talarico Neto, Renata Pontin de Mattos Fortes, Alessandro Rubim Assis, and Júnia Coutinho Anacleto. 2009. Design of communication in multimodal web interfaces. In Proceedings of the 27th ACM international conference on Design of communication (SIGDOC '09). Association for Computing Machinery, New York, NY, USA, 81–88. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/1621995.1622011

[5] María-Jesús Lobo, Emmanuel Pietriga, and Caroline Appert. 2015. An Evaluation of Interactive Map Comparison Techniques. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). Association for Computing Machinery, New York, NY, USA, 3573–3582. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/2702123.2702130

[6] Daniel R. Lanning, Gregory K. Harrell, and Jin Wang. 2014. Dijkstra's algorithm and Google maps. In Proceedings of the 2014 ACM Southeast Regional Conference (ACM SE '14). Association for Computing Machinery, New York, NY, USA, Article 30, 1–3. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/2638404.2638494

[7] Susan Dunnavant. 2010. Create interactive web illustrations with google maps. In Proceedings of the 38th annual ACM SIGUCCS fall conference: navigation and discovery (SIGUCCS '10). Association for Computing Machinery, New York, NY, USA, 267–268. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/1878335.1878401

[8] Karen Church, Joachim Neumann, Mauro Cherubini, and Nuria Oliver. 2010. The "Map Trap"? an evaluation of map versus text-based interfaces for location-based mobile search services. In Proceedings of the 19th international conference on World wide web (WWW '10). Association for Computing Machinery, New York, NY, USA, 261–270. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/1772690.1772718

[9] Davide Pietrobon, Andrew P. Lewis, and Gavin S. Heverly-Coulson. 2019. An Algorithm for Road Closure Detection from Vehicle Probe Data. ACM Trans. Spatial Algorithms Syst. 5, 2, Article 12 (July 2019), 13 pages. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/3325912

[10] Haiquan Wang, Yilin Li, Guoping Liu, Xiang Wen, and Xiaohu Qie. 2019. Accurate Detection of Road Network Anomaly by Understanding Crowd's Driving Strategies from Human Mobility. ACM Trans. Spatial Algorithms Syst. 5, 2, Article 11 (August 2019), 17 pages. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/3325913

[11] Martha E. Pollack. 1995. Evaluating planners, plans, and planning agents. SIGART Bull. 6, 1 (January 1995), 4–7. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/202187.202189

[12] Laurent Denoue, Scott Carter, Andreas Girgensohn, and Matthew Cooper. 2014. Building digital project rooms for web meetings. In Proceedings of the 2014 ACM symposium on Document engineering (DocEng '14). Association for Computing Machinery, New York, NY, USA, 135–138. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/2644866.2644889

[13] Rachida Hassani, Younès El Bouzekri El Idrissi, and Abdellah Abouabdellah. 2018. Digital Project Management in the Era of Digital Transformation: Hybrid Method. In Proceedings of the 2018 International Conference on Software Engineering and Information Management (ICSIM2018). Association for Computing Machinery, New York, NY, USA, 98–103. DOI:https://doi-org.ezproxy2.library.drexel.edu/10.1145/3178461.3178472