Team Pivotal

Summer 2018

Tracker Mission Control

Ble Salia | Sam Sartor | Liam Warfield | Kyoseo Koo

Introduction

Pivotal is a software and services company based in San Francisco. Pivotal Focuses on making Cloud and Agile Development Products. Pivotal's divisions include Pivotal Labs, Pivotal Cloud Foundry, and Pivotal Tracker (based in Denver).

The project was assigned by Pivotal Tracker. The Tracker team is constantly evolving their application to better serve the needs of their users. Tracker maintains a prioritized backlog of project deliverables, broken down into small pieces called stories. It groups these stories into fixed segments of time (called iterations) and predicts progress based on real historical performance (velocity). Our field session project was to track how Pivotal Tracker was being used and show Tracker's global impact. The vision of Mission Control is to come up with answers to the following concerns:

- Does some newly released feature have any impact?
- Where are active users logging in from?
- What new features are they using?
- How many times has a feature been used last 24 hours?

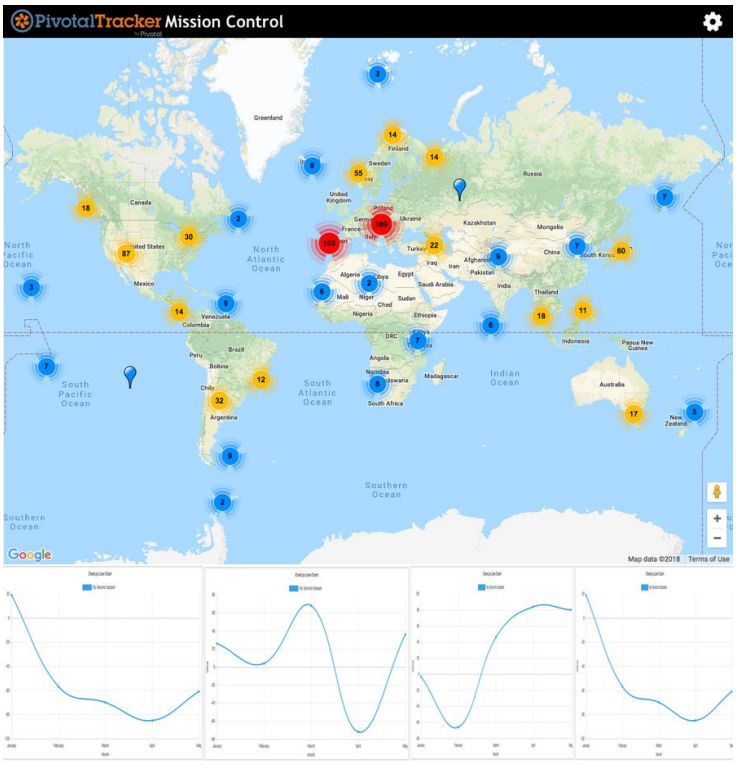

Figure 1: Non-functional mockup of Mission Control, created by the client

Requirements

Functional Requirements

- The web application must show graphs of feature usage over time.
- The web application must show a map of feature usage.
- The map should reflect how much an event occurs.
- There is a settings page to configure tracked events.
- Settings are persistent.
- Features are color coded.
- The webpage must automatically refresh every 30 seconds.
- The application will throw useful error messages:
    - If the client loses internet connection.
    - If the server cannot access feature data.

Non-functional Requirements

- The web application must run in Pivotal Cloud Foundry's PWS hosting service.
    - Pivotal Web Services (PWS) is Pivotal's own "Platform as a Service" product.
- The web application must use event data collected by Mixpanel.
    - Pivotal Tracker uses a service called Mixpanel to track feature use
- The team must use Pivotal's own development process.
    - Test-driven development
    - Pair programming
    - Pivotal Tracker for project management
    - Daily stand-ups
    - Weekly Iteration Planning Meetings (IPMs) and Retrospectives (Retros)
- The web application must receive Mixpanel data through a secure backend.

System Architecture

Mission Control is responsible for displaying the feature usage of Pivotal Tracker. The data is provided by Mixpanel, a service that Pivotal uses to log user interactions as they navigate through the Pivotal Tracker application. This data, however, requires a great deal of processing before it can be visualized with maps and graphs. Additionally, the raw data must be isolated to some secure server, due to confidentiality concerns. Mission Control was broken into two parts, a frontend to display data and a secure backend which communicates over HTTP.

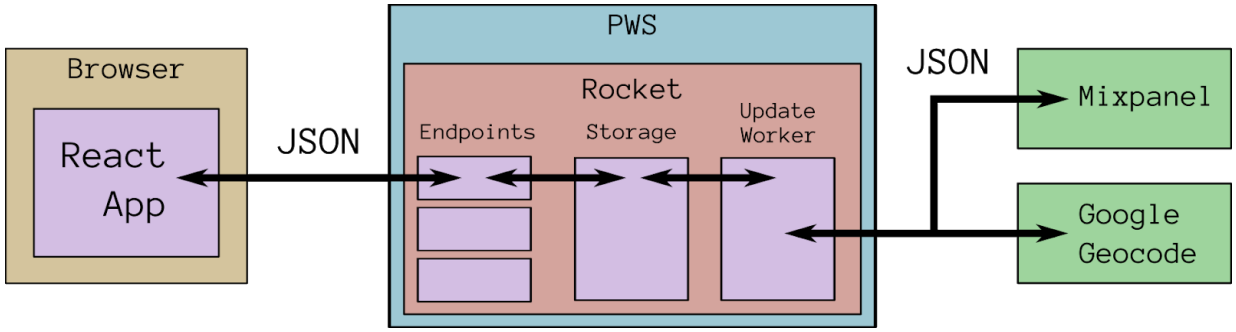

Figure 2: Mission Control Architecture

The frontend is a web application running in the user's browser programmed mostly in JavaScript. It displays the maps and graphs but does not do any data processing. The frontend gets all the preprocessed data from JSON endpoints.

The core of Mission Control is the backend. It is a Rust program running on Pivotal Web Services, a web platform provided by our client. It is powered by the Rocket library for Rust. Rocket handles all requests sent by the browser. Rocket passes each request to our code, which processes the request and replies with JSON files and state data. Mission Control itself periodically loads data from the Mixpanel Export API and Google's Geocode API. This data is then processed and stored inside an internal data store.

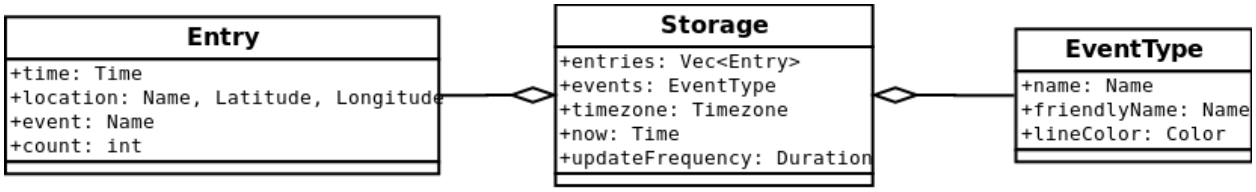

Figure 3: Datastore Schema

This data store acts as an in-memory cache of user interaction data. Since the data storage acts as an initial processing step, it drastically reduces the complexity of the rest

of the Mission Control backend. This allows Mixpanel queries to be run from a separate thread, which can periodically update the data store independent of the rest of the backend. This limits the number of Mixpanel queries to exactly one per minute. Each record from Mixpanel contains four pieces of information: what event the record is tracking (e.g. “sign in”), where this event happened (e.g. “California”), a time stamp, and how many of that event in that time and place occurred. Mixpanel only provides location data as place names, but the Google Maps display requires latitude and longitude. To solve this the server queries the Google Geocode API to associate human-readable names with specific latitudes and longitudes. The storage also retains a cache of this information. Finally, the datastore needs to track timing information, used to implement the automatic data updating.

Technical Design

As mentioned above, the Mission Control backend stores all data received from Mixpanel and Google Geocode in an internal data store. Since this data store might contain millions of entries, each entry must use as little memory as possible. In particular, the same location or Mixpanel event name might appear in thousands of entries. Mission Control would likely have used up all the limited memory available to the backend if it had stored this data in its full, textual form. Additionally, processing entries require lots of filtering on event name, time, and location. Leaving these records as text would have made this process extremely slow since text operations are relatively costly. Therefore, the team implemented a specialized storage system called an interner. Instead of storing event and location names inside each entry, every unique location and event is assigned an integer id. The interners were also used to store metadata for each unique name, such as the longitude and latitude received from Google Geocode. This also prevents costly geographic lookups from ever being performed more than once per location.

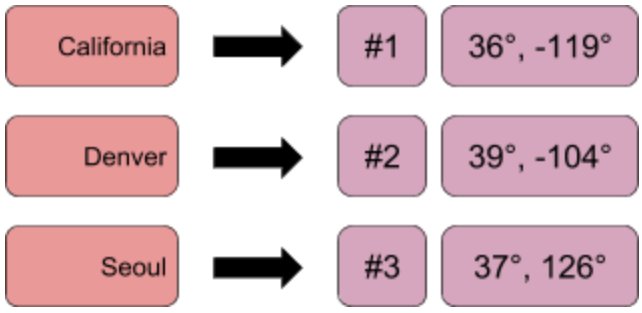

Figure 4: example of how the interner turns strings into integers

Another interesting issue the team ran into was that Mixpanel sends Mission Control records with times in an unspecified time zone. For Pivotal Tracker specifically, this is Pacific Standard Time. Unfortunately, time zones are a purely human concept and do not behave well in the mathematical environment of a computer. Tod make our data more reliable and processable, the Chrono library for Rust was used to convert event times into the UTC time standard, which is designed specifically for use by computers. Chrono, when combined with the stringent Rust type system, required us to correctly handle complex situations such as leap seconds and the daylight savings time switch, but made the code for this logic fairly simple. Interestingly enough, this level of correctness is beyond even the level provided by Mixpanel itself, which states in its documentation that "if [the projects selects] a time zone that uses daylight savings time, there will be minor abnormalities on the transition dates."

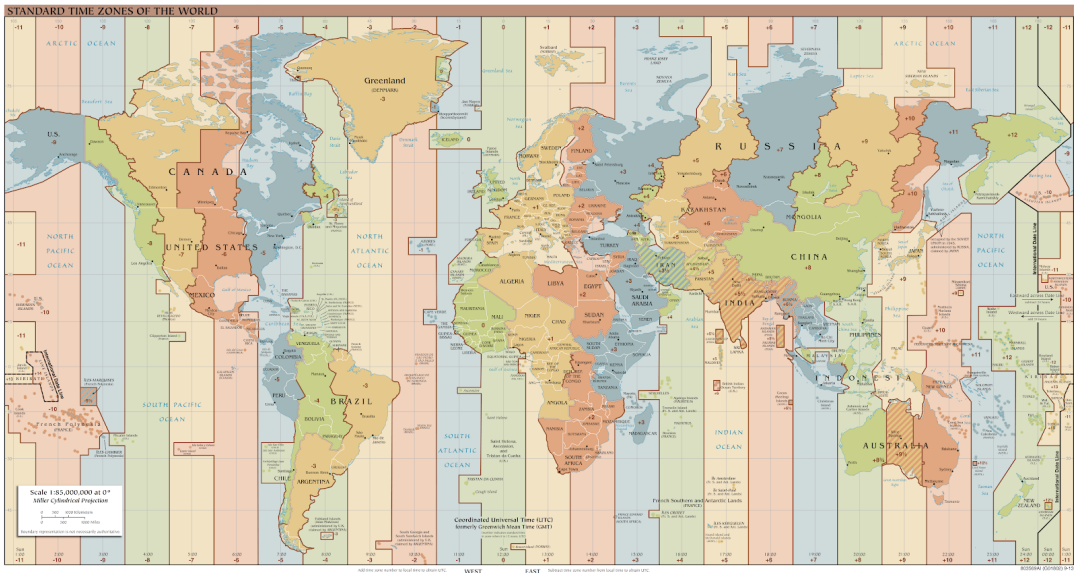

Figure 5: world time zones and UTC offsets, painful to implement from scratch

Design Decisions

Rust

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety. Because the Mission Control backend needs to be robust, multithreaded, and fast, Rust is an ideal choice. The strong type system and declarative approach to data processing entirely eliminated bugs and crashes. In addition, only one of the team members had prior experience with the language, so it provided a better educational opportunity than alternatives such as Python.

Rocket

Rocket is a popular Rust web framework for handling HTTP requests, routing, and responding. One of the project's mentors also has extensive experience with the framework.

JavaScript

Javascript is the primary language for developing in-browser applications. However, several other languages can be transpiled to either JavaScript or WebAssembly. In particular, the Mission Control frontend could have been developed in TypeScript, Elm, or even Rust. JavaScript was chosen because it is ubiquitous and thus important to learn.

React

React is a JavaScript framework which makes for painless user interface creation. It also includes an extension to the JavaScript syntax (called JSX), which can interleave HTML with JS code. React also automatically handles dynamic page updates, which makes live updates to the graph and map reliable and uncomplicated.

Geocode

Mission Control receives location data in a textual form (e.g. "California" or "Stockholm"). These location names need to be resolved as latitude and longitude coordinates for the heat map display. The team could either have used a local lookup database, or some external service. Google's external Geocoding API was chosen because of its. ease of use, reliability, and error correcting (i.e. "Kalifornia" is automatically corrected to "California").

Chrono

Mission Control needed a type-safe way to handle dates and times within the backend. As described in the "Technical Design" section, timezones and time differences

had to be especially robust. Thus, Mission Control imports the `chrono` library for Rust, which enorces correct handling and parsing of dates and times.

Heatmap

While implementing the map display, the Mission Control team and client came to realise that individual pins for each feature and location were not visually intuitive or useful. Several different data visualization techniques were considered instead, and the client decided that a heatmap would be the best option. In addition to the visual improvement, the heatmap library used similar data to the individual pins, so our system did not need to be significantly changed.

No Database

Only a single instance of the backend is needed, which does not ever need to be stopped or restarted, and which uses minimal system resources while running. In particular, Mission Control's cache of Mixpanel data is entirely transient, since it is rebuilt every minute. Thus, Mission Control does not require a persistent database, relying instead on a in-memory datastore. This reduces the complexity of the backend, and the difficulty of learning the project's technology stack. However, integrating the Diesel ORM might be an easy task for the future.

Results

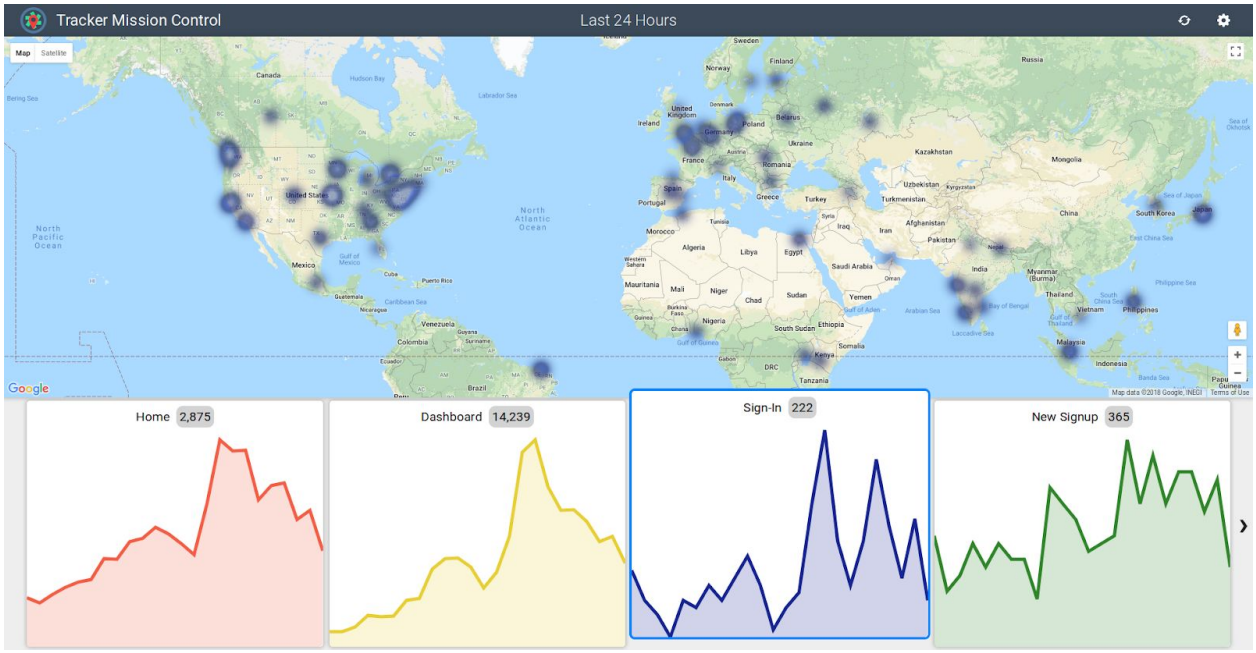

Figure 6: The final product display page

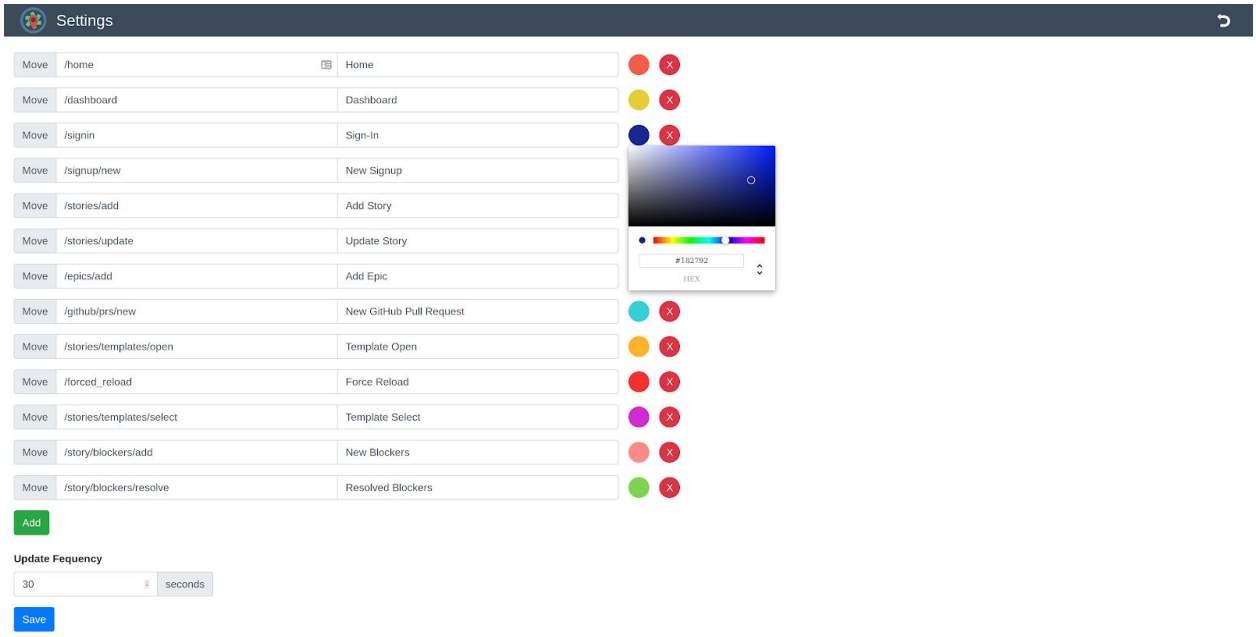

Figure 7: The final product settings page

Tracker Mission Control was a great success, we were able to implement all the client requirements and meet the definition of done. The goal of this project was to create a server that receives information on features of Pivotal Tracker, and then display the data in graphs and a Google Maps heatmap. We succeeded in creating an app that runs efficiently both server side and client side. Mission Control only takes approximately 1 second to load on modern browsers and then updates as new data becomes available without refreshing the page. The server can process over 40,000 entries in only 10 milliseconds and is stable because of our decision to build the backend using the Rust programing language. Also, because Rust enforces explicit error handling, we will never have to worry about unchecked errors.

Mission Control was also a success because it improved our skills as developers. As a team, we practiced full stack development for the first time. We learned how to function as a team in a real work environment. This included communicating regularly with the client over what was desired for each feature. Within the team, pair programming improved significantly as we learned each team member's strengths and weaknesses. We found that letting the less experienced team members drive allowed knowledge to be dispersed across the team. Team members also learned Rust, JavaScript, and their respective frameworks (Rocket and React). Overall each team member experienced significant growth as a software developer.

Appendix

Terms

Pivotal - A software & services company

Pivotal Tracker - Project management software, division within Pivotal, the client

Pivotal Web Services - A platform for developing, managing, and running web applications

Story - A task tracked by Pivotal Tracker

Point - A measure of progress provided by Pivotal Tracker

Mission Control - The software application developed by this team

Mixpanel - A business analytics service that tracks user engagement with Pivotal Tracker

Event - A type of user interaction tracked by Mixpanel, some feature of Pivotal Tracker

Backend - Not directly visible to the user, processes and serves data

Frontend - The visible user interface

Geocoding - The process of resolving a location name to real position

Interning - A method of storing only one copy of each distinct value (also, slave labor)

UTC - The primary time standard by which the world regulates clocks and time

ORM - A tool which provides database access through a language's type system

Product Versions

0.0.1 - Production Deployed Application

0.1 - Mixpanel Visualizations

0.2 - I'm the map, I'm the map, I'm the map

0.3 - In-app configuration

0.4 - It's alive!!!!!! Dynamic data updates

1.0 - Rotating dashboard