Nicholas Carnival
Ryan Suffridge
Lake David
Ross Lannen
06/18/19

Table of Contents

Introduction

Pivotal is a computer software company that focuses on improving software developers work flows. Pivotal makes software to help people who develop software work more efficiently. One of their most popular products is the Pivotal Tracker. Pivotal tracker is a way for team to manage how they develop software using agile methods. Part of having a better workflow is knowing how a team is performing and knowing how each person on that team feels about a project. The way the Pivotal does this is through health checks,. Pivotal wanted a better way to do team health checks. These are meetings in which a project team get together to discuss the working environment and the project. This is done with a list of categories and questions that a facilitator asks participants about. This is done with a white board and an excel document to keep track of previous health checks in order to compare the overall trend of the teams health throughout the lifetime of the project. In order to simplify and automate this, the team decided to create a web application that will be a collective of mobile and desktop participants. The health check is facilitated by one person, and everyone else is the participant. This requires two different views for the web app. In order to keep track of each user, the team chose to use people's google accounts, because everyone at Pivotal has a google account. This is done through the google authenticator.

After logging in, users (facilitator and participant) are brought to a 'teams' page. This allows users to create teams and interact with the questions and categories of a team. This view also has a redirect to the history page which shows information on all of the past health checks. The user is able to start and join health checks from this page. This page, as well as all other pages are optimized for both desktop and mobile view. This is not done just by having a responsive web design, but also by having different features for both views. After starting a health check, the facilitator and the viewer will be brought to the first question in the first category and the participants will vote on how they feel about a topic. Each question is a team health based topic that also has an

overall trend. This trend is about the overall change in a topic in order to determine if the team is improving, getting worse, or staying the same. For example: If the question asked “How satisfied are you with your working environment?”, the example options would be Happy, Crappy, or Mixed. Once a user has chosen how they feel about this, they will be brought to a discussions page. This page shows how everyone voted and allows the facilitator to lead a discussion and take notes. The facilitator can then navigate to the next question. After all of the questions have been answered, the users are brought to a summary page that shows how each question was answered as well as the trend of each question.

Requirements

In order to meet the deadline, the team had to plan out every feature and detail. This was done using the Pivotal Tracker where the team kept track of every feature that needed to be implemented as well as who was working on those features and how many of them were completed each week. Requirements were broken down into Functional and Nonfunctional requirements.

Functional

Functionally this project needed to make team health checks easier and more fun. The client wanted this to be done through a web application that supports both mobile and desktop. This includes both the front end and the back end. The front end would be what the users interact with, and the back end is where all of the data is stored. User authentication had to be as easy as possible to use, and allow users to quickly log in and out of the system when using it on a public computer. The system also has to handle this information securely so that their private information is not leaked. In order to keep the users information secure, the team chose to authenticate through google. The web app has to update each user’s device when the facilitator decides to advance to the next question. This web app also has to have separate views for both desktop and mobile users. These two views are the facilitator and the participant view.

Non Functional

Initial authentication is accomplished through google OAuth, allowing users to use their existing Google accounts to complete a health check. Subsequent requests

are authenticated through a user session stored in memory on the server and accessed using a signed cookie with the session id. This makes signing in as painless as possible for the user and removes the issues that plague password authentication systems.

Non Functionally, the team chose to use React for the majority of the codebase. This is because Pivotal uses React and because React is extremely popular with a ton of free documentation. React is a JavaScript framework that simplifies making a web application. React focuses on user interaction, which is required when creating a team health check. The majority of this application revolves around users interacting with the site. React was used for the front end, and Node JS was used for the back end along with Express JS as a routing library. Knex JS and Objection JS were used to communicate between the Node server process and the database. This website is hosted on Pivotal Web Services because the client is Pivotal, and the software is easy to work with. Pivotal Web Services hosts the front-end client application, the back-end web API, and the database which makes it easy to have interaction between the site and the database. The team chose to use NodeJS because having all of the codebase written in JavaScript was a benefit to moving quickly, and working along the full stack. Pivotal also has many experienced JavaScript developers, so handing off the app to the client to maintain into the future is as painless as possible. Node JS is directly supported by Pivotal Web Services through buildpacks, giving easy server configuration right out of the box. Material UI was used for all of the icons, fonts, and buttons. The team chose to use Material UI because it is the most popular UI framework for React and has easy-to-use documentation. Easy-to-read documentation is a must have for a project this size and this time limit. In order to have JavaScript that interacts with a SQL database, the team had to use a Relational Query Builder, more specifically ObjectionJS. This tool allows JavaScript code to build SQL queries which could then be sent to the PostgreSQL (PSQL) database hosted on Pivotal Web Services. PSQL was chosen because the team is most familiar with this flavor of SQL. Pivotal makes it easy to implement a PSQL database into their cloud services through ElephantSQL, a hosting service for PSQL.

System Architecture & Technical Design

The team health tracker is split into three main components: the front-end client application, the back-end API server, and the data persistence storage. A diagram of these 3 pieces is shown in Figure 1.

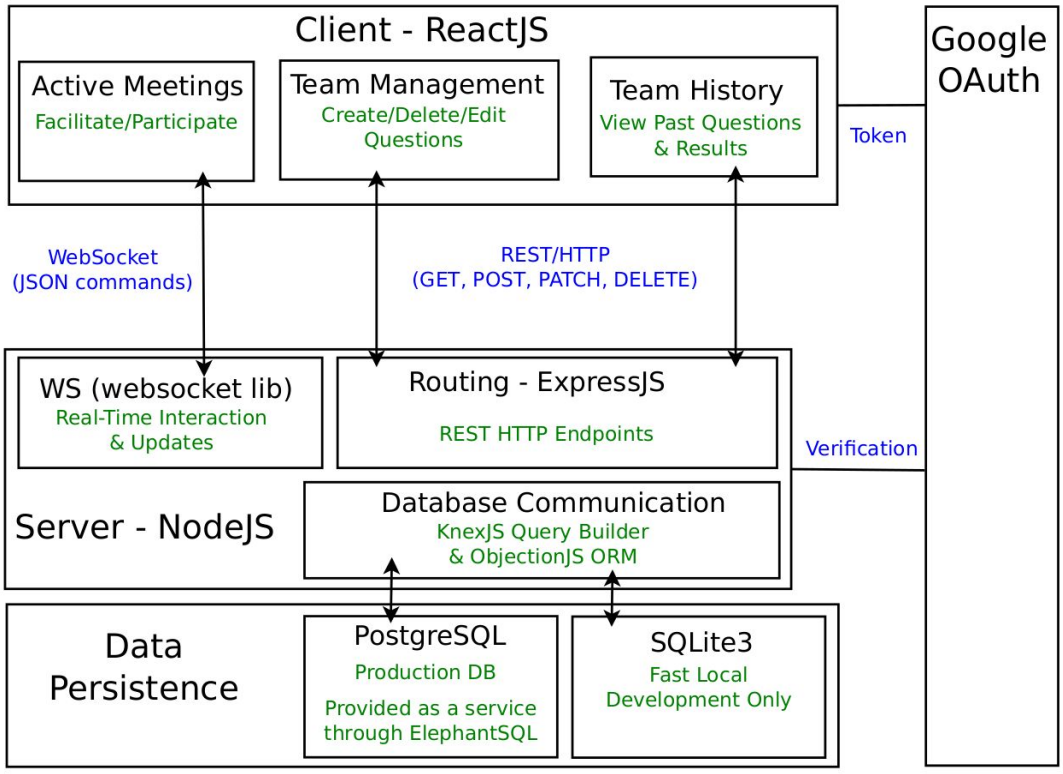
The front end is built as a client rendered Single Page Application (SPA). This means that the front end files for any one of the pages is a single piece of HTML, CSS, and JavaScript that is served up on any request to the app server. Once fetched, the application parses the URL and renders the “page” that the user navigated to, and fetches the data necessary to display on that page from the server. Subsequent navigations within the app do not cause the frontend to be reloaded, but merely re-rendered by the JavaScript and optionally fetch just the data it needs. This makes the app feel more responsive and interactive, as if it was a native desktop application. Some browsers can even cache the static front end files, making the overall network usage of the app extremely low for multiple visits.

Major dependencies used on the client include React for page rendering, Material UI for the components and styling, and Axios for the server communication. The browser native websocket APIs are also used for the more interactive pages.

The back end API is written in JavaScript, and runs on NodeJS. It uses Express as its routing library to create the API endpoints that the client app communicates with. The backend presents two different types of endpoints.

The first are the HTTP REST endpoints that allow for the team management, and basic CRUD (create, read, update, delete) operations on teams and their questions. These are used in the team management and history portions of the app. These follow REST best practices as closely as possible and use the different HTTP verbs to interact with resources and collections of resources within the application. The system architecture is shown below.

Figure 1: System architecture diagram



The second endpoints are the websocket endpoints. There are two of these available, one for participants and one for facilitators. These allow facilitators to start meetings, and for participants to join meetings once started. Once a socket connection has been established, that client is subscribed to any updates that happen to the meeting in progress such as someone joining, leaving, voting, or otherwise interacting with the app. Along with subscribing to updates, the participants and facilitators can take actions in the form of a small set of commands that can be sent to the server over the socket. For the facilitator, these commands include advancing through the steps of the healthcheck, and creating discussion notes for any of the questions. For the participant, these commands include voting on a question, and voting on the trend pattern of a question.

The third component of the system is the data persistence layer. This is provided to use as a service through ElephantSQL, a hosting service for PostgreSQL relational databases. This service is automatically linked to the API server through Pivotal Web Services service system.

The database schema is specially made to allow history of past meetings to be preserved. When a team is managed, behind the scenes it modifies questions and categories on an unstarted health-check entity. This entity is implicitly created whenever a team is created, or whenever a meeting is ended. This allows for teams to modify questions and categories for future meetings, while maintaining results, discussions, and trends for all past questions. It also presents itself in a way that is hidden from a user, and allows them to intuitively interact with the software on a team level. The entire database schema is shown in Figure 2, showing this and all other entity relations.

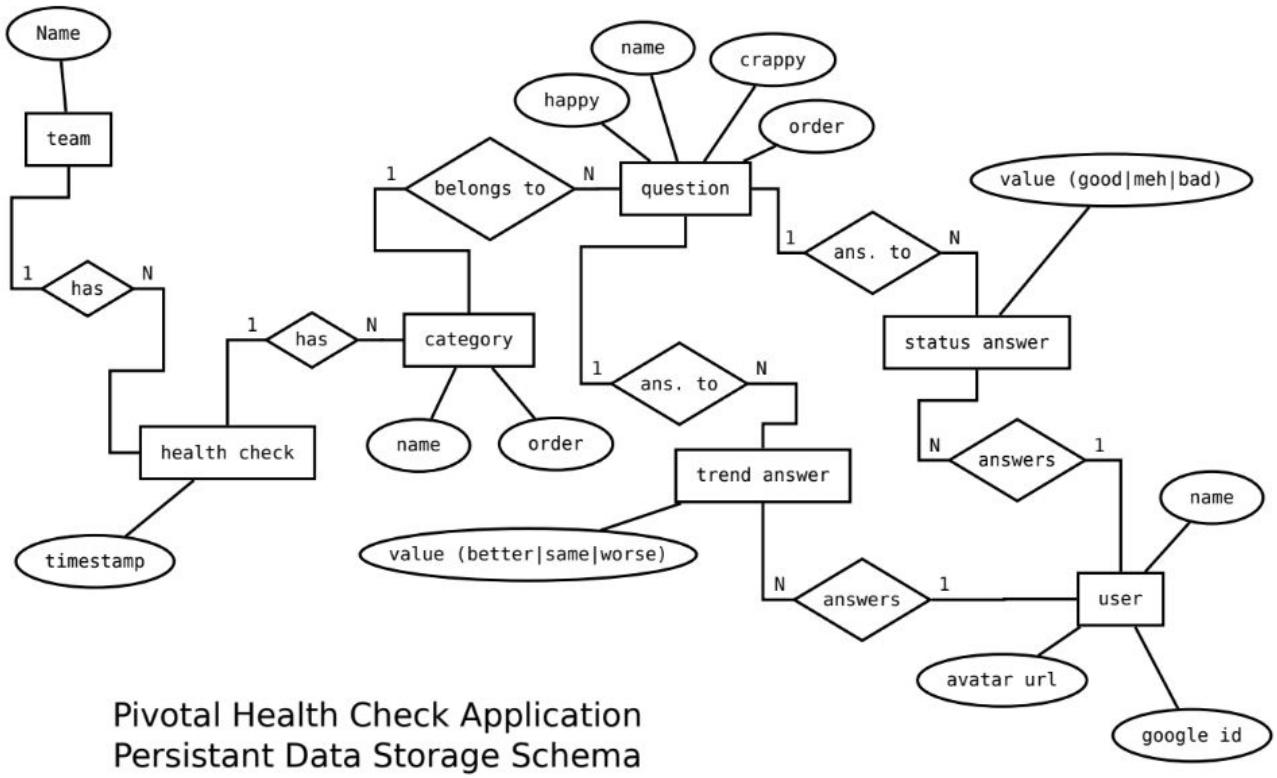


Figure 2: Entity Relationship Diagram

To interact with the database, two Node libraries were used. The first of these is KnexJS. Knex is a JavaScript query builder. It provides a database agnostic way to construct SQL queries through javascript functions and objects, that can be executed on any SQL database. Using Knex, it is very easy to specify different databases for development, testing, staging, and production. For the app, the team chose to have different databases assigned for local development and cloud hosting. Knex is also in charge of managing the database migrations. These migrations are specified in a set of files in the project that create the database DDL and schema. Every time the server is started, it checks whether all migrations have been applied to the database. If there are unrun migrations, the server automatically applies them, modifying data as needed, before starting the server. This allows for deployments with database changes to be more easily automated when deploying to PWS.

The second library used for database interactions is ObjectionJS. Objection is an Object Relational Mapper (ORM) that allows a higher level of abstraction over database records and tables than is available with raw SQL. It is built upon Knex, and adds many features such as being able to define models, validate JSON inputs when creating new model instances, and easily create complicated queries with many levels of nested joining and fetching. For example being able to fetch a team along with all of its current

categories, questions, and answers to those questions becomes much easier when using Objection as all these relations can be defined and chained together.

Quality Assurance (QA)

Definition of Done:

In order to maintain a high level of quality, the team chose to have a thought out “definition of done”. The team’s definition of done focuses mainly on what the client wants from us. The most obvious definition of done is finishing the project on time with the features that Pivotal wants. However, this is only half of the definition of done. Quality is also a large part of the plan. Pivotal will most likely move forward with this project, so everything that the team gives to the Pivotal must live up to their expectations and standards. The team’s definition of done is when the product meets the clients needs and meets the following quality assurance guidelines.

Agile:

The main focus in the team’s quality guideline is to make sure that every group member is not only contributing to the project, but contributing in a way that is both useful and of high quality. In order to do this, the team is utilizing the agile framework. This framework promotes solving problems as a collective rather than having individuals solve problems. This helps with the quality of the code written because multiple people see the product before it is published, reducing bugs and sloppy code.

SOLID:

Using solid principles while developing software allows for changes, while not having to rewrite or change old code. This is crucial because the client may want to make changes to the product after most of the product has already been developed. SOLID principles allow for changes to be made at any point in production, without harming any of the already functional feature. All code should be open for extension, but closed for modification before being merged to the production branch.

User Acceptance/Interface Testing:

The user interface of this project is manually tested. The team chose not to spend time learning how to automate interface tests because the team thought that it would take too much time to learn. The team was more focused on finishing the product, and if given more time, tests would be implemented to improve quality and performance. Before any changes to the web application reach the client, at least three team members have to see the changes and agree that it maintains the teams level of

quality. After the team has agreed that the changes maintain this quality, the changes are reviewed by the client. The client can either accept or reject these changes, leaving all the large decisions up to the client. This allows the team to only contribute features that the client wants.

Code Reviews:

Whenever a new feature is completed, the code is sent to another team member for review. The reviewer is chosen based off who wants to review the code, as long as it isn't the person who wrote the code. The reviewer can decide whether or not the code should be sent to production or not, and if so, the reviewer will leave comments on the portions of code that do not uphold the teams level of quality.

Program Analysis:

At the beginning of this project, the team created a style guide that all code would follow. Initially, this was done manually, but now the team uses a code formatter. PrettierJS is a consistent formatter that runs every time a team member saves code in their editor. This forces all of the code written to look the same, making it easier to read any snippet of code in the project. The team also utilizes linters. These linters analyze source code to flag any errors or bugs. An example of this in JavaScript would be an unused import. Using a linter, the user would be warned of this. The linter does not make any changes to the code because there are cases in which it recommends deleting code that is important. The team is left to decide whether or not they will follow the linter's suggestions. Another form of analysis that the team uses is paired programming. Two people are more likely to spot issues when compared to one.

Version Control:

The team is using version control in order to make it easy for everyone to work on the project without interfering with what someone else is doing. The team is using GitHub to handle version control because it is what the team is most familiar with.

QA Overview:

Most of the team's quality decisions are based off what the team is most familiar with. This is because the team chose to spend more time learning how to develop web products rather than learning web development analysis tools. By using what the team is already familiar with, the team saves time which allows for a faster completion of the final project. This allows the team time to fix any issues that come up once everything is "done".

Results

The goal of this project was to create a better format to do a team health check. Specifically the goal was to have a facilitator use a web app to guide a team through questions and discussions with both mobile and desktop support. This health check needed to have a way for the facilitator to make both questions and categories that the team can answer, as well as the history of previous health checks. The client wanted a way to monitor the long term health of its teams in a fun and exciting way.

Unimplemented Features:
There were a couple of features that did not get finished due to time constraints. The team was not able to implement membership to each team. The web app does not keep track of who is on each team, and anyone can join every healthcheck. This is not problematic because the web app will only be accessed by Pivotal employees. Membership would allow for only specific people to create teams and it would allow for more privacy among team health checks.

Performance Tests:
This project is currently working in Safari, Firefox, Chrome, and Opera. The mobile version of the project works on Safari and Chrome. Using the built in performance testing tools of Chrome, the team was able to test the speed of every site interaction. Creating a team takes 19.33 ms, going to the edit a team page takes 21.8 ms, starting a team health check takes 15 ms, advancing to the next question in a health check takes 8ms. Another tool that was used to test the speed of the website was keycdn, a free online web analytic tool. On average it took 1s to connect to the web app from Dallas Texas. The web app does not have any resource extensive features, so the team decided that no further performance tests were necessary.

Usability Test:
The team tested every iteration of the project during and after developing it, according to agile practices. Included in all of this was the testing by the clients. After any progress was made with the project, an employee of Pivotal would review the changes and make sure that is was acceptable. This ensured that the client received exactly the product that Pivotal wanted. Once the final iteration of the project was finished, the team sought after outside testing. The test was given to employed people in order to test the flow of a health check. The only complaints given by the test-takers was the questions were too obscure. This is not an issue because the questions are user defined. The user tests

were given with the default questions, which are ambiguous. The users experienced no lag or any bugs with the software, Hooray!

Future Work:
This project will be maintained by Pivotal. The people at Pivotal plan to use this software for their own team health checks. The team that worked on this project does not plan to work on this project after the class ends. Pivotal will maintain all aspects of this project and likely make changes based on what their employees think of the health checks. Pivotal will most likely add Role Based Displays. This feature was not implemented by the team and was requested only if the team had extra time to implement it. This feature will change how the health checks are displayed based on if the current user is a facilitator or a participant.

Lessons Learned:
Overall, the team learned how to develop a web application. Going into this project, the team had little to no experience with web development, and by the end of the project, the team had developed a web application that will be used by a multi-million dollar company. The team learned how to create a front end UI using the React framework, and a backend using NodeJS. The team learned how to develop as Full Stack Engineers. The team gained a better understanding of how the Agile workflow helps a project. By using Agile, the team was able to create a product in 6 weeks, with no prior knowledge of the tools that were going to be used. The issues faced while trying to work cross platform taught the team how to change their workflow to suit a team environment, rather than working as an individual. This project allowed the team to explore the real world side of software engineering.