The Regis Company

Consolidated Operations Data Viewing Web-App

Summer Field Session 2019

Adam Beziou
James Brant
Jared Lincenberg
Jordan Newport
Advisor: Donna Bodeau

Contents

List of Figures

1 Introduction

The Regis Company builds, sells, and maintains online training simulations, providing hosting and support services for these simulations. When a simulation experiences an issue or goes down, the company must respond promptly to resolve the issue.

However, the company's response time is restricted by a lack of readily accessible information on clients and their simulations, as the data on each client and simulation is not consolidated by any single service and support staff have to hunt around on different servers for information. The company tasked us with creating a tool for consolidating all of this information and displaying it in an easy-to-read format for support staff.

Our response was Regis Dash, an online dashboard. The primary goal for this project was to eliminate this bottleneck by creating a webapp that provides a real-time overview of the simulations maintained by the Regis Company. The dashboard provides visualizations of data on all currently running simulations, such as pie charts of uptime, as well as provide more detailed data, including URLs, locations of repositories, and contact information, for individual simulations. The real-time nature of this dashboard allows it to update displayed information as new information becomes available without refreshing the page. In addition, the dashboard can be used to edit information for each site and add new sites for any client. We also created a testing suite for automatically checking the information of simulations and updating their information in the database.

2 Requirements

2.1 Functional Requirements

1. Login and Authentication
    (a) Provide user authentication through Google to access the website
2. Client View
    (a) Provide a view of all simulations for every client or one specific client
    (b) Provide a side navigation bar to select different clients
3. Simulation View
    (a) Provide a view of all relevant details related to a single simulation
    (b) This view shows different parts of the data in 3 different components:
        i. A visualization and graphing component
        ii. A list component
        iii. A ”big table” component
4. Health of simulation View
    (a) Provide a visualization of the up and down times in a graph and table
5. All of these views are dynamically updated when new information is available on the backend

2.2 Non-Functional Requirements

1. Does not leak any sensitive company or client data to non-admin users
2. Restricts access to the dashboard to users with verified Regis Company (@regiscompany.com) emails
3. Requires that users sign-in with a Google Account through Google's Authentication system; there is no other authentication option
4. The website is written in JavaScript with React and run on a Node.js server
5. The website's repository is maintained by the Regis Company on Gitlab

2.3 Use Cases/Stories

The main use case for the application is a client support engineer or on-call developer, who has two main needs when providing support to Regis Company customers. The first need is to be able to monitor all running simulations to see if any are ongoing problems. The second need is to be able to see the details of any particular application on short notice so that when a support call is placed, the support engineer can respond quickly.

Another potential use case is tracking projects and past work. Since technology is constantly progressing, the methods used for developing simulations will change, and being able to refer back to older simulations made for the company will keep them more consistent with past simulations in design and methodology.

3 System Architecture

We architected 3 components that were necessary to make the website meet the functional requirements: the back-end component, a front-end component, and a data management component.

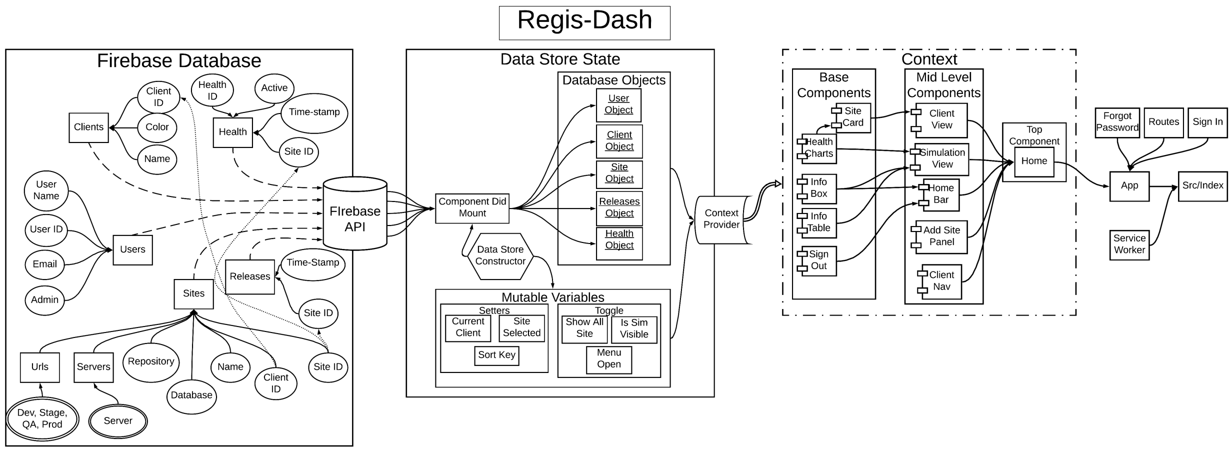


Figure 1: Data and component flow between different parts of the Web App.

The back-end component consists of a Firebase server that stores of the client and site information. We used Firebase as the database because it can automatically push out updated information and stores our data in a JSON (JavaScript object notation) format, providing easy compatibility with JavaScript.

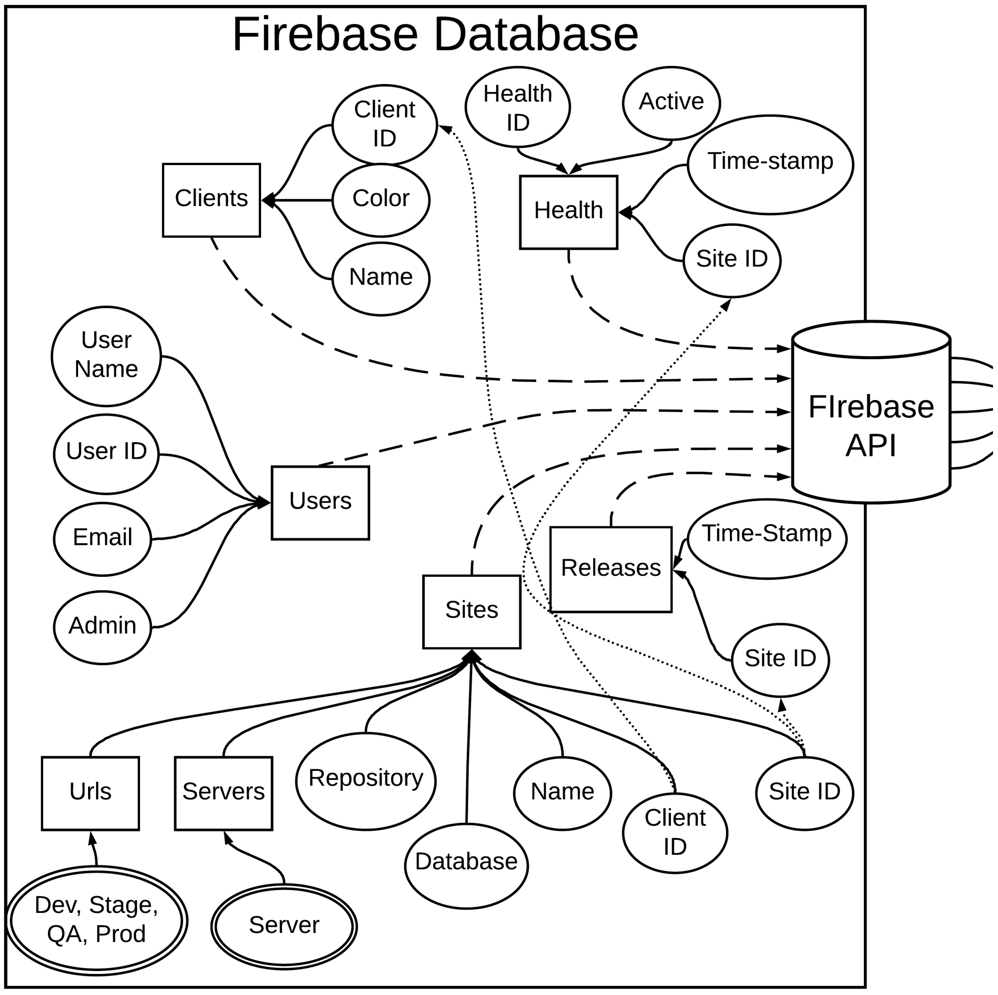


Figure 2: An approximate layout of data in the Firebase Database. The arrows point the parent of the children nodes. Dashed lines show that element is accessed by the Web App. Dotted lines mean that the elements are a key to other datasets.

The front-end component sorts and displays all of the information as a card-based dashboard that is hosted online. We used HTML, CSS, and JavaScript to develop the front-end.

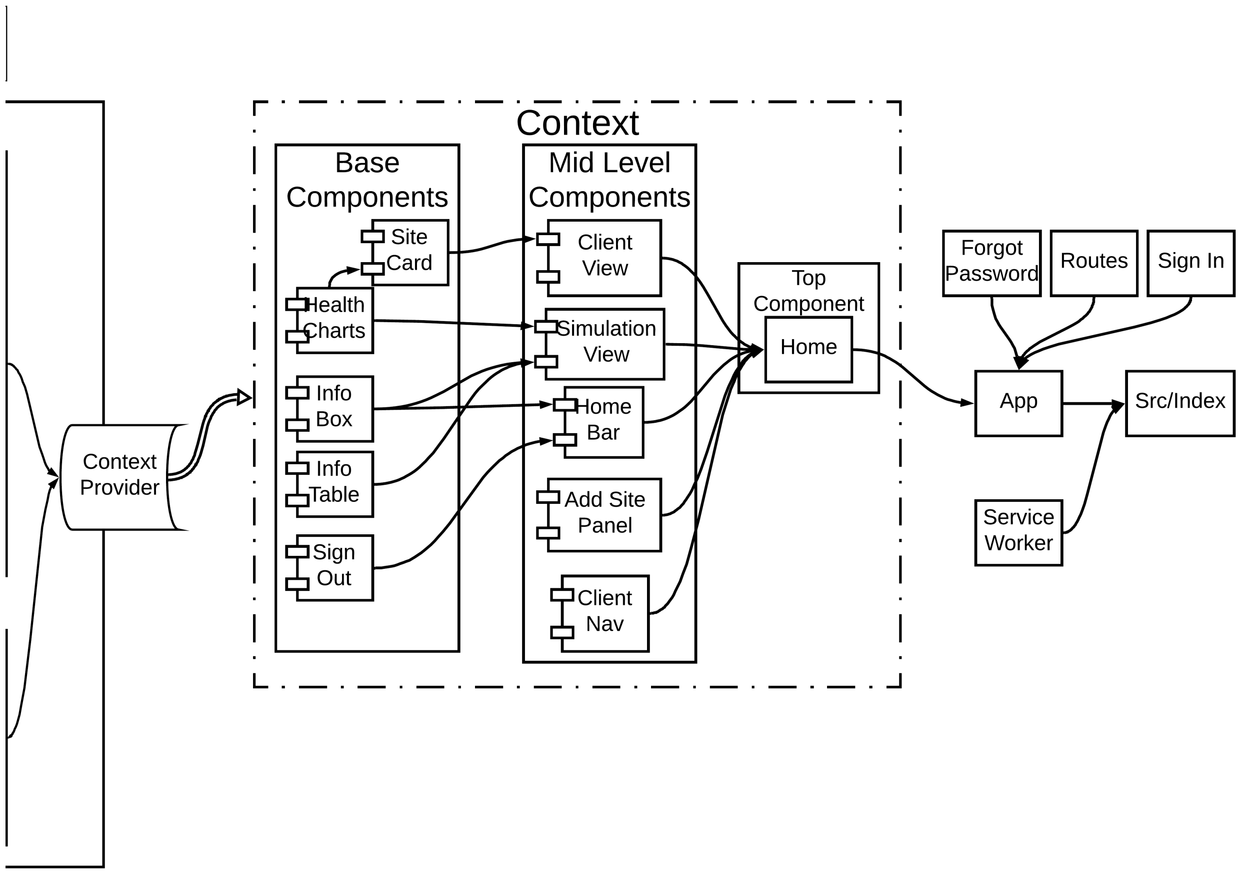


Figure 3: The context keeps track of the state of each of the components on the right. The components return JSX (HTML) elements that are composed together to form the webpage.

The data management component acts as a middle-man between the back-end and front-end by getting data from the back-end, manipulating the data, and passing it to the appropriate parts of the front-end. We used the React JavaScript framework to build the data management component.

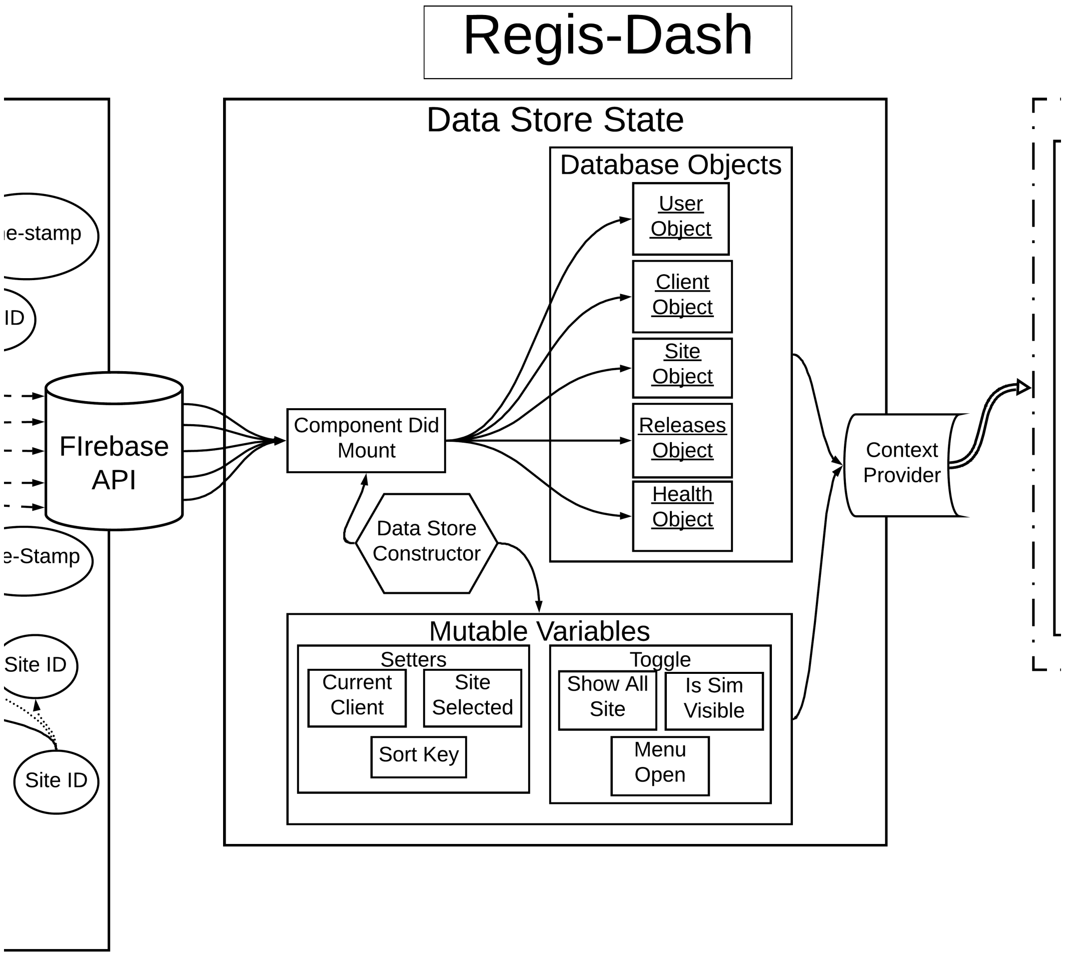


Figure 4: DataStore is a pseudo-global state object which gets information from the back-end server and provides information to the front-end components.

4 Technical Design

4.1 React and Firebase

We are using React, a JavaScript library for building user interfaces that can update asynchronously and in real-time using React Components that manage the state of the application and render JSX elements. React can be used to create web pages that can "react" to incoming data in an OOP way using classes. We need the dashboard to update in real-time as new information comes in from IT support and simulation websites, but we also need the backend to be as flexible with real-time data in order to synchronize with React. We built the backend using Firebase, a real-time cloud-hosted database that utilizes data synchronization—every time data changes, any connected device receives that update within milliseconds. Therefore, when any change in data occurs in Firebase, the update is sent to React Components which then updates the web app automatically.

4.2 Selenium Testing

We are using the Python bindings for a web automation framework called Selenium in order to generate our health data. Selenium allows us to run a "web driver" executable in order to automate tasks on a website. This means that we can automate checking whether a website is up, by writing a script to visit the website, log in, and check to see whether a couple of expected elements are present. This allows us to know whether the website is fully functioning or if it is down or otherwise broken.

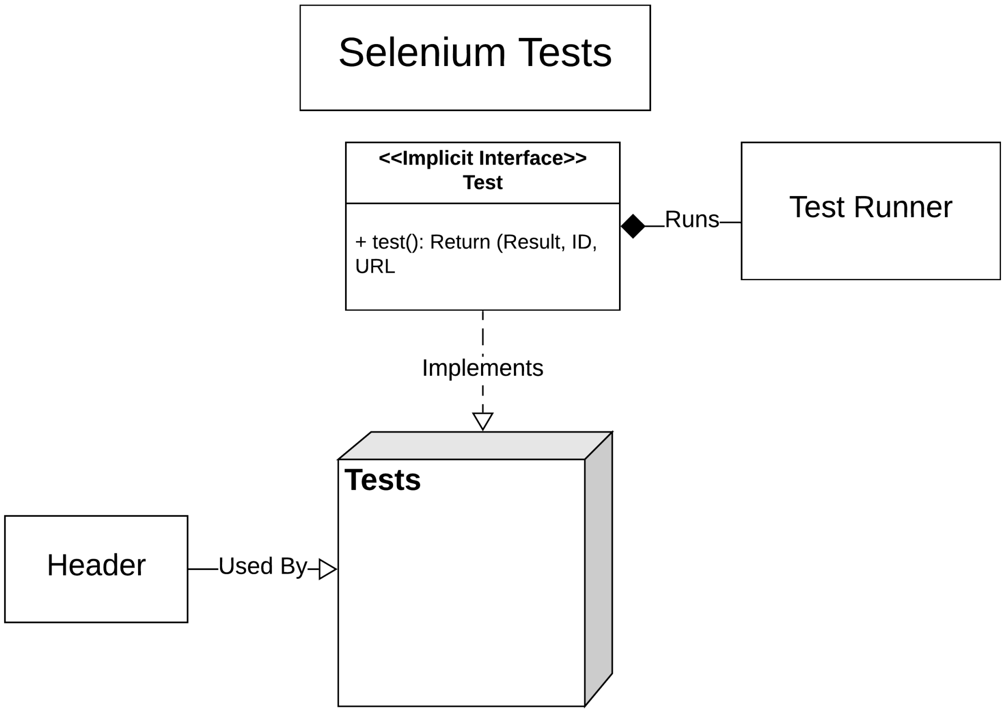


Figure 5: Architecture of the Selenium testing setup

Our architecture (5) makes it simple to write a new test by moving all the boilerplate out to a header file and leaving just the parts that change per test. This makes it much faster to write a test or update a test when the site it is written for changes, which will make it easier for The Regis Company to use our framework to keep track of uptime on all its sites, even after we leave. Once tests are written, the runner just imports the test function from each test file, then aggregates the results to write back to the database. An example test is included in appendix Section 2.1 to demonstrate how simple we have made this for the user–all a user has to do is modify the URL, ID, and test function itself.

There are also some sites that do not have a full Selenium test. For these sites, our test runner runs a much simpler test, where it just makes an HTTP request to the website and checks what that HTTP request returns. This enables us to check whether the website is up or down, ensuring that we have at least some coverage for any site that is in the database, regardless of whether it has a full test written.

5 Quality Assurance

5.1 Agile/Scrum

We have strictly adhered to the Agile and Scrum process to improve our response time to changing requirements and coordinate efforts between team members. We planned weekly sprints on Mondays where we made individual plans for the week and had daily evening stand-ups where each member listed what they did during the day and stated their plan for the next day. The proceedings of these meetings and the progress we have made on each task were recorded in a shared Trello board. Our Scrum master and client, Stephen Unger, holds each of us accountable for our chosen tasks and records our progress in the board. This process keeps our team highly unified and collaborative by keeping each team member informed of the progress of every other member and the project as a whole. It also allows each team member to give constructive feedback and participate in the decisions made on other sections of the codebase, which results in a more cohesive and error-free product.

5.2 Testing

We performed various types of testing on our code and our product as a whole, which has allowed us to detect errors and make changes without worrying about breaking things or making bad decisions. These types of testing include uptime testing and user acceptance testing, in addition to manual testing.

Uptime Testing

We are using Selenium testing to regularly ping our production site and all of the simulations then record the results in our Firebase server. This data is used to display uptime/downtime charts on our dashboard. This can be used not only as a diagnostic tool but to catch and track issues early that may be occurring on a server.

User Acceptance Testing

We have been in continuous communication with our client as part of the agile methodology. We have discussed most major changes and revisions of the code base with them in order to keep with their vision. By seeking approval for code and design changes we have ensured that the website is well suited for their purposes.

Manual Interface Testing

We manually tested out all features of the website, as writing selenium tests for every new and rapidly changing feature proved to be too time consuming and impossible to keep up to date. We initially planned on using Jest to automate testing of our JavaScript code, but we ran out of time before any tests could be made. The rapid pace and unstable nature of our development cycle made writing tests almost impossible. Writing tests for functions, classes, or elements that change on an hour-to-hour basis and might not exist in a few days was pointless, so we stuck to manual testing for all of the features on the website.

5.3 Code Reviews

Our code has continuously been reviewed by different members of the Regis Company as progress is made. Our team lead reviews changes to the codebase during our daily stand-up meetings and other engineers and managers give feedback on the code and interface as the product is shown around the company. We also reviewed each others' code as each member has had the opportunity to work on and review multiple sections of the codebase. Our continuous review process helped us to catch issues early before they have a chance to become ingrained in the codebase and identify future "problem areas" where we could improve or might have issues in the future

6 Results

6.1 Successfully Implemented

We successfully implemented several features, including almost all of our functional requirements, as listed below:

1. Login and Authentication: We have successfully implemented secure Google sign-in.
2. Client View: We have provided a view of all simulations for every client or one specific client, in addition to a ”hamburger menu” to select different clients. An image of this is provided in Section 1 of the Appendix.
3. Simulation View: We have implemented a view of all relevant details for a particular simlulation, including three separate components. An image of this is An image of this is provided in Section 1 of the Appendix.
4. Health of Simulation View: We have implemented a simple visualization of health data in the form of a pie chart. We were unable to implement anything more complex.
5. Views do successfully dynamically update when new information becomes available on the backend.

6.2 Missing Features

We did not implement most of the “Big Table” component, integration with Visual Studio Team Services, and detailed interface testing. These features were not implemented due to either lack of time or overwhelming technical difficulty. Given more time or development experience, all of these features could have been implemented.

The Big Table component, the bottom table in the Simulation View, was intended to provide a more comprehensive view of the information pertaining to a single simulation. The table is functional, but all of its tabs do not display any data. Most of the information that would have been displayed, such as users, paths, and sessions, was not given or inaccessible. In addition, some of the information that we had and could be displayed in the table, such as health logs, were formatted in a way that made sequentially displaying them in a table extremely difficult.

Visual Studio Team Services (VSTS) integration was hindered by the lack of compatibility between our simulations’ identifying information and VSTS’ information. VSTS provided information on simulations, including release information, but the simulations in our database did not have the same identification numbers, names, or URLs as the simulations in VSTS, even if they were technically the same simulation. This incompatibility made integrating VSTS into our dashboard impossible without first modifying the information in VSTS to fit our own.

Detailed interface testing of our own website was not implemented, as we discovered that any change to the layout, even one that did not actually change the functionality in any way, broke the test because of the fragile hierarchical nature of the DOM. Because of this, we stuck to manual testing, to reduce the hassle of updating the test frequently.

6.3 Browser Testing

We viewed our website on Firefox, Chrome, Chromium, and Safari. It performed as expected on all of these browsers, with any discrepancies in appearance merely cosmetic. The one noticeable discrepancy is that a long link in a variable-width element wraps in Firefox, but instead widens the element in Chromium and Safari. This is not an important difference, as the website is perfectly usable either way.

We did not test our website in Internet Explorer or Microsoft Edge, as we were informed that these are not important use cases for our client, who will likely only use the Chrome browser. We also did not test on mobile, as that is also not one of our primary targeted use cases.

In addition to browser testing, we also performed usability testing and manual interface testing.

6.4 Usability Testing

Our team leads, Tyler Messenger and Stephen Unger, tested the software throughout development. They are both support staff, so getting their approval was highly important. As they tested the site on a variety of devices, they requested features to enhance the usability of the website. We changed the color of text over non-white backgrounds to white with a black border and modified the layout and size of the InfoBox component based on Tyler's recommendations. This is because Tyler had concerns that the website would otherwise be unusuable by people with visual impairment , particularly color blind people. Stephen leads us through the development process and added dozens of features to our weekly sprints as he tested our changes every day.

6.5 Future Work and Possible Extensions

All of the features that are described in the above "Missing Features" section could be implemented if given enough time and access to more company information.

Some other potential future features include:

- Mobile compatibility. We would like to make the website more functional on phones or tablets, as the website is currently too wide for phones, which produce a disappointing user experience.
- Records of previous issues that the simulations have had.
- Calendars and schedules of current and future sessions of simulations so the support team can prepare.
- A way to link to specific views of the website to share via email or messaging between support staff.
- Proactive alerts. When downtime is detected on a site, notify the support engineer immediately by email, rather than waiting for the engineer to notice themselves.
- A mechanism to have the simulations and the servers they run on monitor their own health and send the data to our webapp to improve the collection of health data.

6.6 Lessons Learned

Over the course of this project, which was all of our first time working in industry, we learned several important lessons that will make us more effective on large-scale projects in the future.

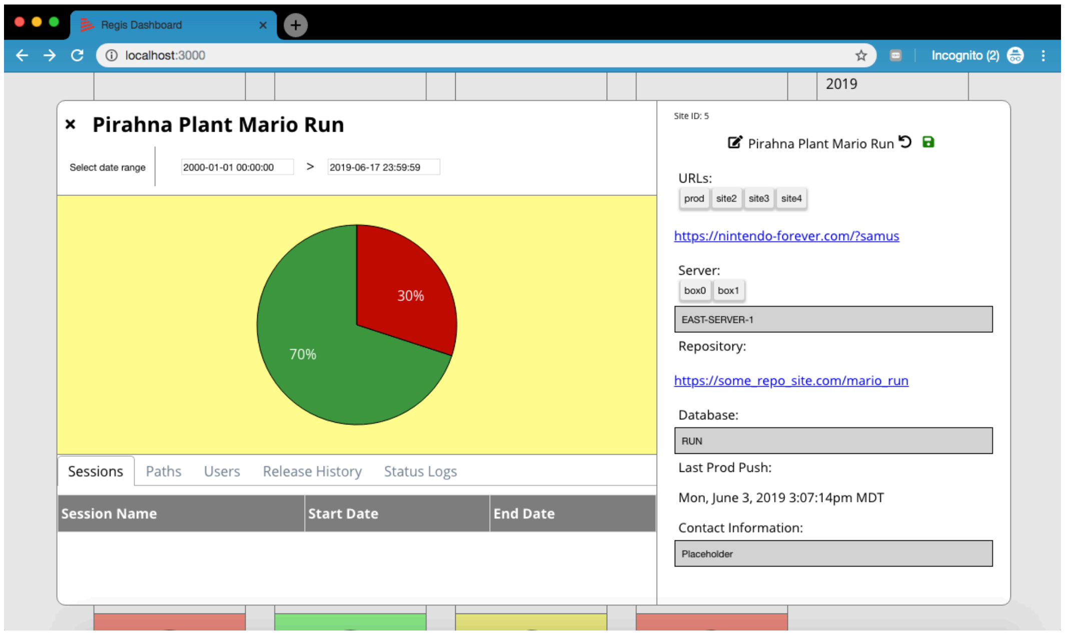
- The communication about structure needs to be explained and explored with the team continuously. By each of us focusing on specific parts of the website we were able to quickly make significant progress but when others were tasked on updating or change these feature it could drastically slow down progress.
- It is very important to have a testing environment separate from the production environment. We overwrote a production database because of a malformed update and additionally struggled to debug errors that only showed up on "real" URLs on the open Web since the pipeline to push code onto the production URL was slow.
- The premature optimization of features slows development time significantly while not guaranteeing any benefits; in future to make a feature, we should first make it work, then make it correctly, then make it fast.
- Accessibility is an important concern, as the interface needs to be readable and functional for all users. This manifests in even seemingly unimportant things like color choices. Understanding this takes time, testing and communication to get right.

6.7 Conclusion

Overall, the project was a success. We learned and had a lot of fun with HTML, CSS, and JavaScript, as well as interacting with back-end servers. Our final product satisfied company requirements and our client contacts with only a few minor features missing. The product was shown off to the company and will be integrated into the support staff toolkit and continue to be worked on as part of a company-wide initiative to improve support and DevOps. The product has room to grow as there are many potential features that can be added and its scope can be expanded beyond just serving support staff. In addition, some members of the support staff have already used the tool. The project ended up satisfying the requirements and fulfilling its purpose as well as being adopted into the company.

Appendices

1 Screenshots of Website



Session Name	Start Date	End Date

Site ID: 5

Pirahna Plant Mario Run

URLs: prod site2 site3 site4

https://nintendo-forever.com/?samus

Server: box0 box1

EAST-SERVER-1

Repository:

https://some_repo_site.com/mario_run

Database:

RUN

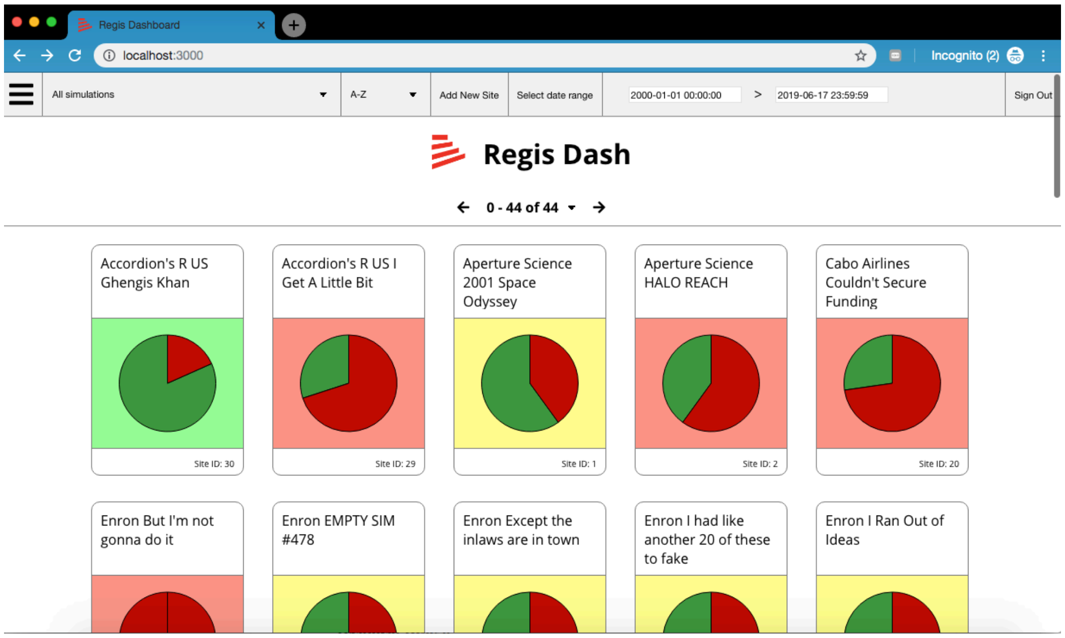
Last Prod Push:

Mon, June 3, 2019 3:07:14pm MDT

Contact Information:

Placeholder

Figure 6: Details on one particular simulation



Regis Dashboard
localhost:3000
Incognito (2)
All simulations
A-Z
Add New Site
Select date range
2000-01-01 00:00:00
>
2019-06-17 23:59:59
Sign Out
Regis Dash
0 - 44 of 44
Accordion's R US Ghengis Khan
Site ID: 30
Accordion's R US I Get A Little Bit
Site ID: 29
Aperture Science 2001 Space Odyssey
Site ID: 1
Aperture Science HALO REACH
Site ID: 2
Cabo Airlines Couldn't Secure Funding
Site ID: 20
Enron But I'm not gonna do it
Enron EMPTY SIM #478
Enron Except the inlaws are in town
Enron I had like another 20 of these to fake
Enron I Ran Out of Ideas

Figure 7: Overview of all simulations

2 Code

2.1 Selenium Test

We have included a snippet of code from one of our selenium tests (the one targeting our own website):

```
#!/usr/bin/python3
import test_header
import time

def test():
    url = 'https://regis-dash.firebaseapp.com'
    test = test_header.Test(url, True) # True to run headless
    id = -2

    def test_function():
        # login as administrator
        test.wait_css('.user-pass-form')
        test.driver.find_element_by_css_selector(
                '.user-pass-form > input:nth-child(1)').send_keys('example@example.com')
        test.driver.find_element_by_css_selector(
                '.user-pass-form > input:nth-child(2)').send_keys('Pa$$w0rd')
        time.sleep(1)
        test.driver.find_element_by_css_selector(
                '.user-pass-form > button:nth-child(3)').click()

        # make sure there is a first item in the burger menu
        test.wait_css('#home-content > div > i')
        test.driver.find_element_by_css_selector('#home-content > div > i').click()
        test.wait_css('button.bm-item:nth-child(1)')
        test.driver.find_element_by_css_selector('button.bm-item:nth-child(1)').click()

    test.function = test_function
    return (test.run(), id, url)
```