VISOIL – DRILLING DATA GRAPHICAL DISPLAY

Project Report

Oroshi IT
Thomas Llewellyn
Jeong Seok (Jason) Kim
Shkumbin (Ben) Muli

VISOIL – DRILLING DATA GRAPHICAL DISPLAY

Submitted By:

Jeong Seok Kim, Shkumbin Muli, Thomas Llewellyn

In partial fulfillment for the award of

Diploma

Information Technology Computer Systems

School of Information and Communication Technology

Southern Alberta Institute of Technology

July 2020

VISOIL – DRILLING DATA GRAPHICAL DISPLAY

Submitted By:

Jeong Seok Kim, Shkumbin Muli, Thomas Llewellyn

In partial fulfillment for the award of

Diploma

Information Technology Computer Systems

School of Information and Communication Technology

Southern Alberta Institute of Technology

July 2020

Approval:

Supervisor:

Name: _____

Signature: _____

Academic Chair:

Name: _____

Signature:

_____

Abstract

Directional drilling operations produce a lot of data, which needs to be carefully captured and recorded by geologists who format it into useful graphs and reports. Traditional methods of manually creating these drilling graphs are time consuming and costly, and the end product is typically a static spreadsheet that does not offer any additional functionality or value to the data presented. Visoil aims to provide a solution to this problem by automating the data download and graph generation processes, through the use of a web application, and providing interactive features that give added-value to the drilling reports.

The Visoil – Drilling Data Graphical Display project was originally designed to be implemented from a local web server with VPN access and other local hardware resources, but due to a sudden change in world events, the project needed to be moved to an alternate location. In the end, the project was implemented through the use of cloud services, with minimal hardware and networking resources being utilized.

After overcoming initial major changes to the implementation of the project, Visoil managed to successfully deliver an application that reads real-time drilling data from a sample repository of data, and proves that a fully-operational web application of this type is feasible. The implications of this project are that exploration and production companies would no longer need to rely on static spreadsheets and graphs in order to get their drilling data, but can utilize powerful and inexpensive cloud resources to access their data in real-time.

Acknowledgements

The Visoil project team would like to acknowledge the people and organizations whose contributions and support were instrumental to the project's success. Firstly, we would like to thank our project sponsor and client, Dino Zelantini, for allowing us to pursue this opportunity, and for providing us with constant feedback and support along the way, even when it meant long meetings after hours and collaborating remotely due to the pandemic. Next, the team would like to thank the Southern Alberta Institute of Technology – School of Information and Communication Technology for the use of its facilities and resources, and for enabling us to succeed even when faced with the loss of resources. Thirdly, we would like to thank our instructors, Dr. Banks Odole, and Chris Driedger, for their guidance and continued support throughout the lifetime of the project, their help ensured that we didn't loose sight of our goals, and kept us motivated through times of adversity. Lastly, we would like to thank our peers and families for enduring alongside us while we worked to accomplish our project goals.

Table of Contents

Table of Figures

1. Introduction

1.1. Project Aim

The aim of this project is to develop a web application that will demonstrate a proof-of-concept for automating the process of creating graphical displays for real-time geological drilling data.

1.2. Intended Audience/Beneficiaries

The intended beneficiaries of this project are exploration and production companies that spend time manually creating drilling reports to use for directional drilling guidance and final drilling reports.

1.3. Key Stakeholders

Project Sponsor: Dino Zelantini, Odole Banks

Project Customer: Dino Zelantini

Project Team Members: Jeong (Jason) Seok Kim, Shkumbin (Ben) Muli, Thomas Llewellyn

1.4. Project Scope

This project involves building a fully functional web app that displays real-time drilling data in the form of a line graph. The project work includes designing and building both the front-end user interface, as well as the back-end database and server functions that support it. The front-end website will include the following features:

Feature	Description
Login Page	A login page that authenticates the user and creates a session on the webserver
Wellsite Selection Page	A wellsite selection page that allows the user to select which wellsite data to view
Graphical Display Page	The live display of the selected drilling data
Logout Confirmation Page	A page to confirm when a user has successfully logged out of the application

Of these, the primary focus of the app will be the graphical display, which will have the following core functionalities:

Functionality	Description
Automatic Updates (“Live Data”)	- A live display of the drilling data pertaining to the site that was selected, which is updated automatically on 30minute intervals (1800s)
Line Graph	- This data represented as a colored line graph of subsea elevation (y-axis) vs measured depth (x-axis), and will include the following data points: o ROP (Rate of penetration) o Gamma o Gas o Well Path
Layer Toggles	- Toggle switches to turn data layers on or off
Annotation Tools	- Annotation tools to add textual notes, flags, and tags to the graph
Printing Option	- Print to pdf option that will print either the entire graph or a selected area of the graph to pdf
Email Option	- Option to email the printed pdf using the client’s installed email application

1.5. Not Included in Product Scope

The following features are not included in the project scope

- Displaying data that is not in the same format as the test data
- Displaying data or values other than those specified in this document
- Non-website applications (i.e. desktop applications)

1.6. Project Approach

The VisOil project was completed by three IT Computer Systems students, working over a 4-month period to plan, design, develop and implement a web application. The team members used free code editors to locally develop the user interface and the back-end systems. Git was utilized to manage version control, Google’s Firebase platform was used to provide the real-time database, user authentication, and cloud functions. Firebase is a mobile and web application development cloud platform that makes the building, testing and deployment of apps easier for developers [1]. For production, the website was hosted with Firebase Hosting to ensure that it was accessible for uses to experience in a real-time environment. The application was delivered in a series of releases to ensure that productivity was maximized and to streamline functionality testing. The following image outlines the major release versions of Visoil.

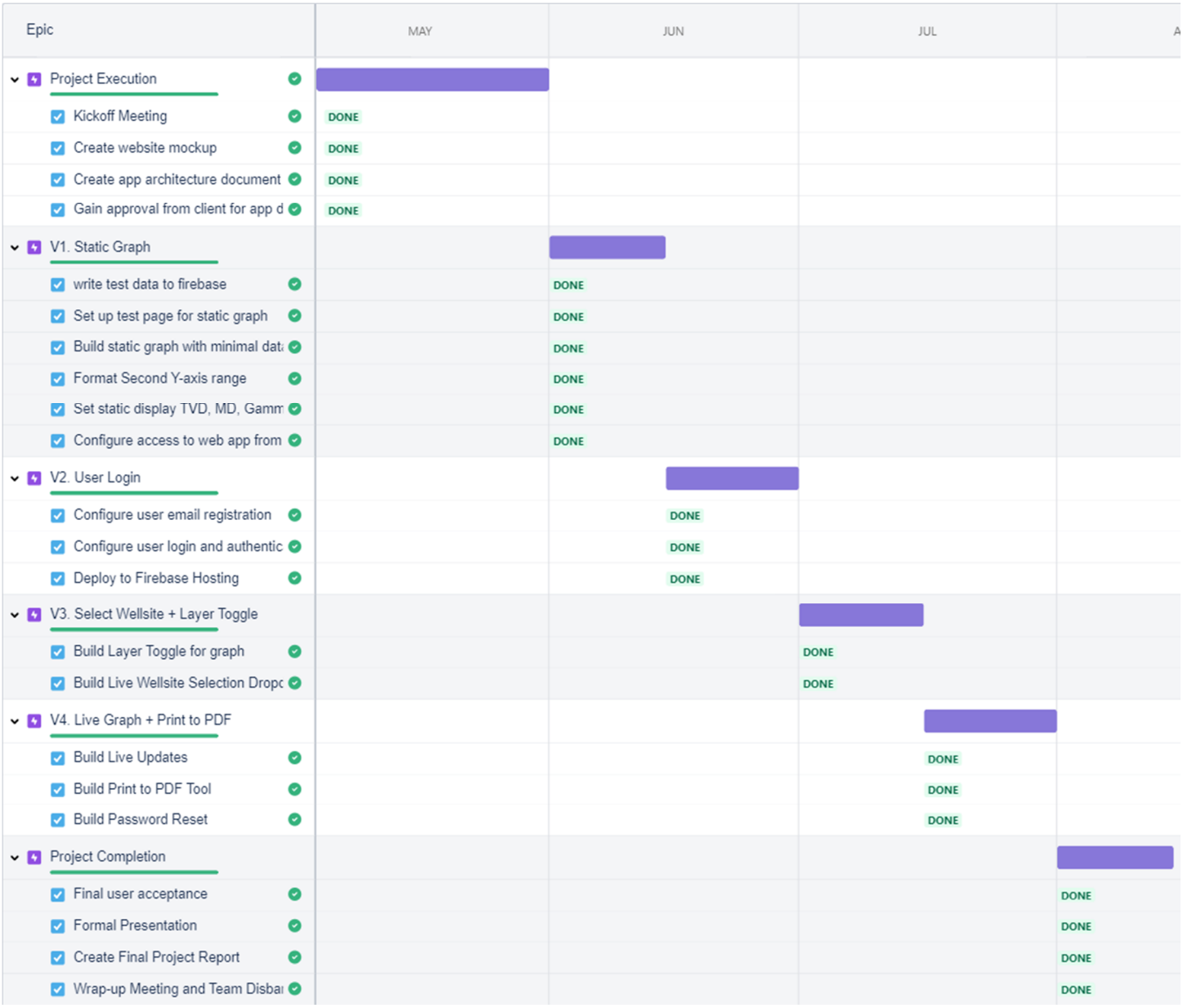


Figure 1-Visoil Version Release & Project Schedule

Each team member focused on a key functionality of the current release and collaborated over MS Teams and GitHub to coordinate tasks and identify and resolve issues.

1.7. Assumptions

It was assumed that all data samples needed to build a compatible application would be supplied by the Client, and that at no time will the unfinished application be tested in a live environment where it could cause damage. It was also assumed that the project team possessed the technical skills and knowledge necessary for working on the project, and that enough time has been given to complete the project. The final application would only be able to translate data given in the specific format that is used in the current environment, and any changes to the data format will result in a need to re-design the application. Finally, it was assumed that the application would be delivered only as a proof of concept, and any further implementation of a functional web app in a production environment was not part of the project.

2. Background

2.1. Context

Dino (the Client) works for petroleum exploration and production company that provides detailed graphical reports of the wells that they are drilling. These data for these logs are generated by gamma ray sensors on the drilling rig, that measure rock formations as it is drilling. This information is sent wirelessly to a third-party database, where the Client logs in and downloads the data as a spreadsheet of raw data points. This data is then translated manually into a line graph utilizing custom made excel spreadsheets. Currently it takes up to 4 hours to download new data and translate it into a graph.
This graph is then used to relay information to the drilling operators to provide guidance on which direction to steer the drill, in order to avoid 'bad' sectors that could damage the drill. These graphs are generated periodically during the drilling, and the Client must choose when to log in to the database to get updated information, and the entire process must be repeated.
Sometimes a request is made from the drilling site to get an update on the data, and must wait for the graph process to be completed before they can receive a response
After drilling is completed, the accumulated data is formatted into a final report that is printed to paper and presented to engineers who will use the information for fracking operations.

The process of manually creating well logs is time consuming and inefficient. This can be overcome by implementing a tool to generate these graphs automatically from the raw data that is produced by the drill.

The Visoil - Drilling Data Graphical Display project will include the design, development and implementation of a web application that automatically download, format, and present the wellsite data as graphs.

2.2. Problem Statement

The Client's current process of making graphical reports manually is inefficient and incurs heavy labor costs the Client's company. This process also introduces unnecessary risks to the security of the data being manipulated and minimizes the value of the final reports that are produced.

2.3. Constraints on the Approach

There is a strict time-constraint on the project, as it must be delivered by August 2020, and the project team will be working with limited human and financial resources. Therefore, the scope of the project must be kept strictly within the constraints of the schedule and budget, which will reduce the number of added-value features that can be included in the final product.

The project was originally intended to be implemented in a lab environment with physical severs acting as the back-end infrastructure, with custom-build networking and security features. Due to a global pandemic that resulted in a loss of physical space to implement the project, the team requested that it be moved to a cloud deployment (*See "Appendix B Request for Change #001" document for full details.).* Various cloud platforms were considered, and Google Cloud's Firebase was eventually chosen.

2.4. Methods and Tools

The front end of the application was developed using HTML, CSS and JavaScript, while the back end was implemented on Firebase (from Google Cloud Services), using Firebase Storage, Firebase Real-Time Database, and Firebase Hosting. No additional equipment or subscription costs were incurred for the duration of the project, and all work was completed within the constraints of a free-tier trial for Firebase. Version control and collaboration was done through Git/GitHub, and various online meeting applications were used for meetings and reviews. All team members worked remotely from their homes to complete the project, as there was no access to public facilities or workspaces during the time of project.

3. Specification and Design

3.1. User Interface

The following images are of the front-end user interface of the web app:

3.1.1. User Login

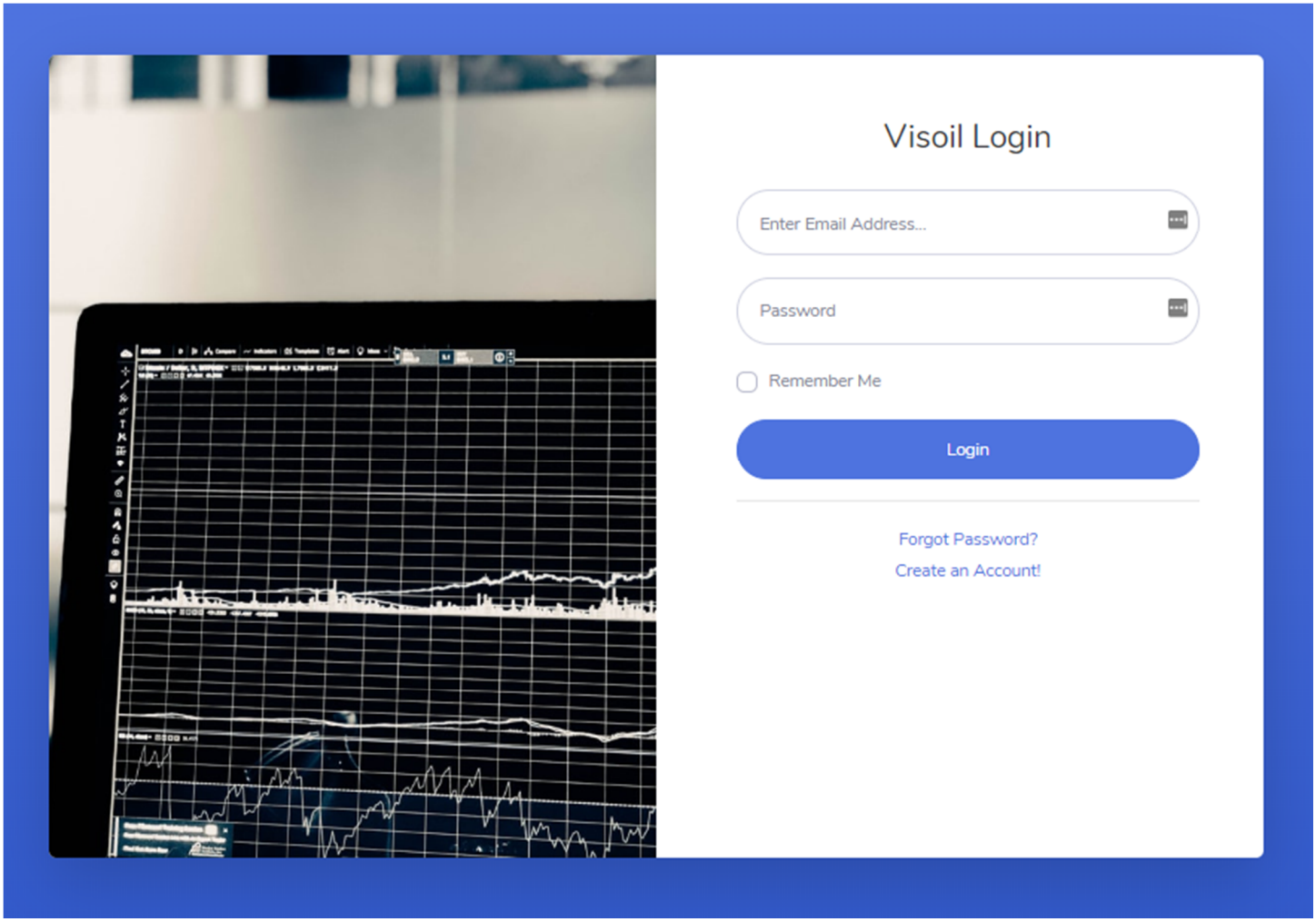


Figure 2 - Login Page

3.1.2. User Registration

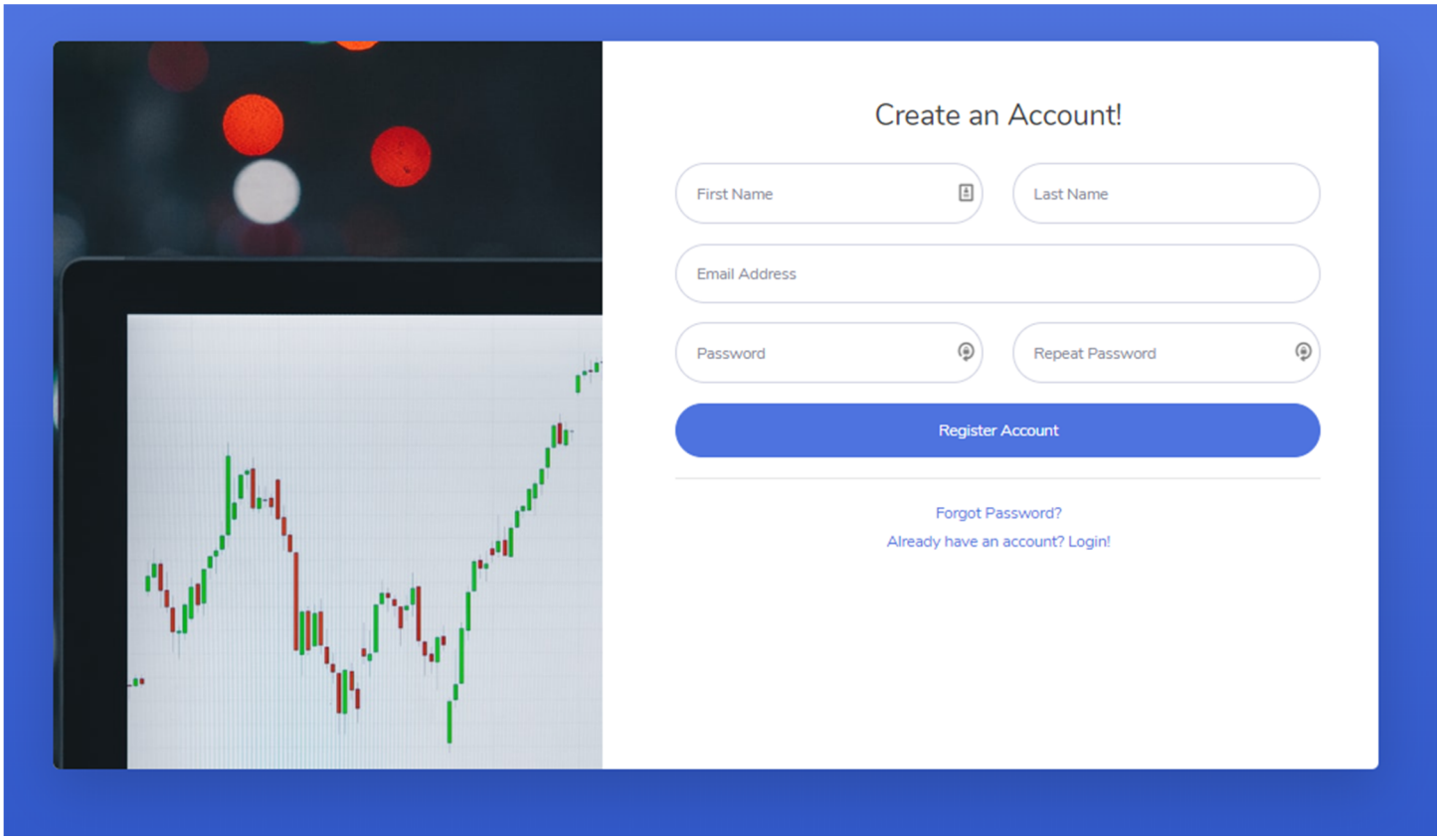


Figure 3 - Create Account Page

3.1.3. Forgot Password

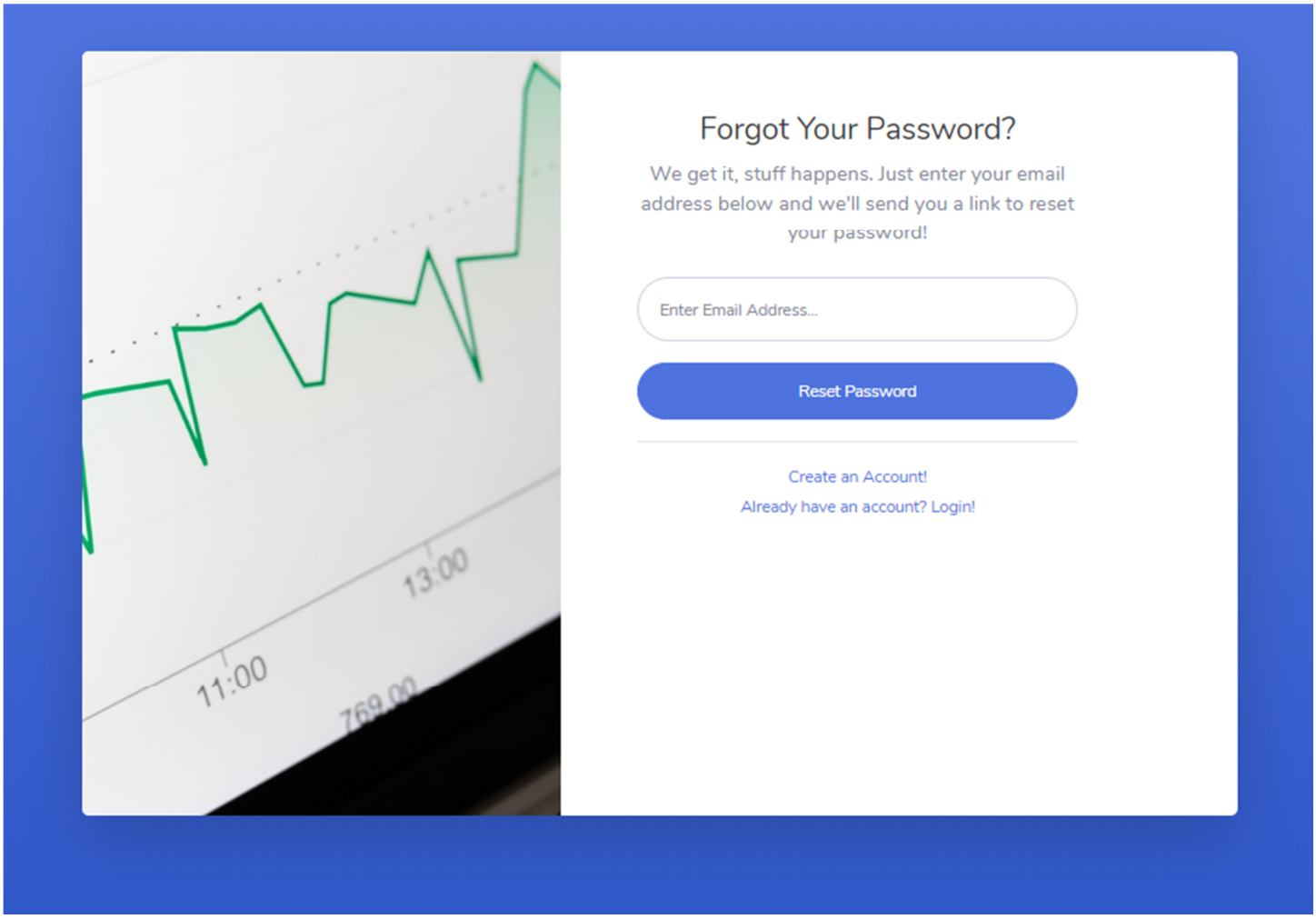


Figure 4 – Forgot Password Page

3.1.4. Dashboard – Wellsite Selection & Static Graph

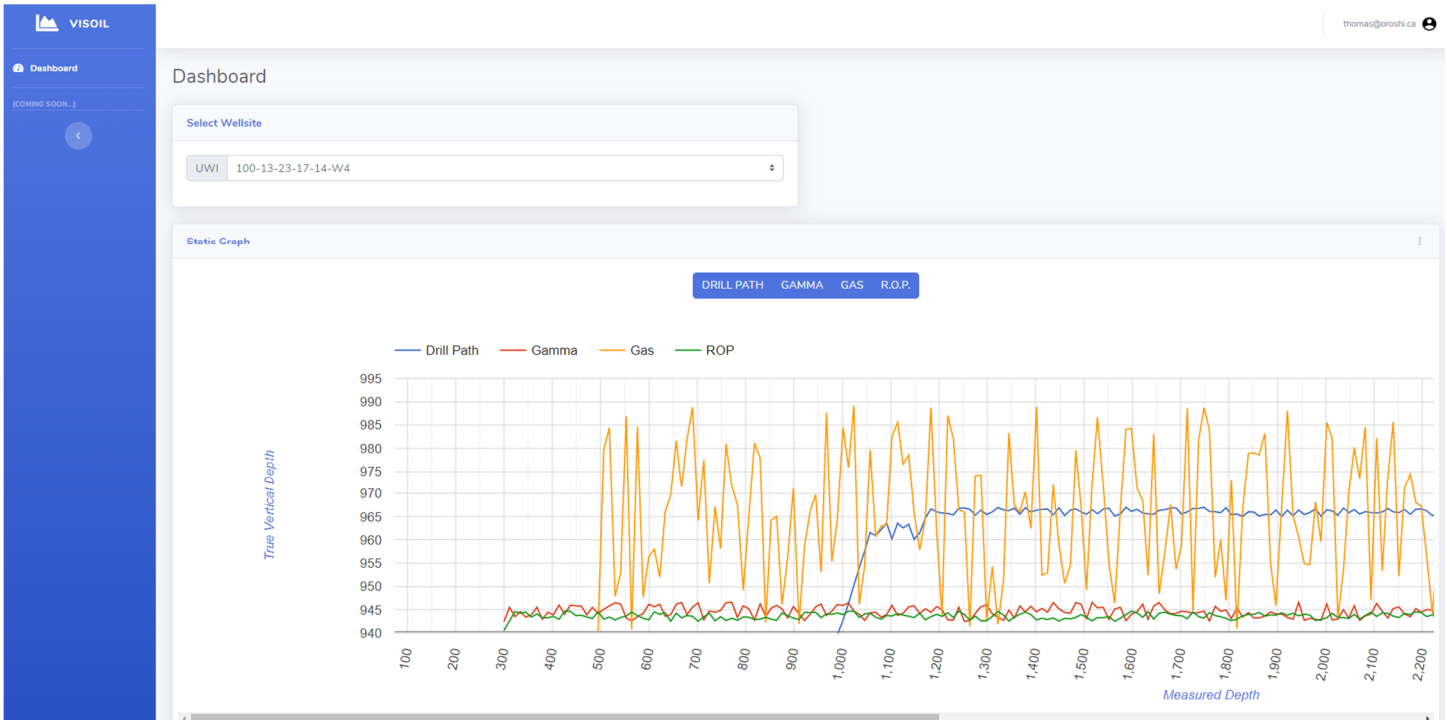


Figure 5 – Dashboard Page, Wellsite Selection & Static Graph

3.1.5. Dashboard – Live Graph

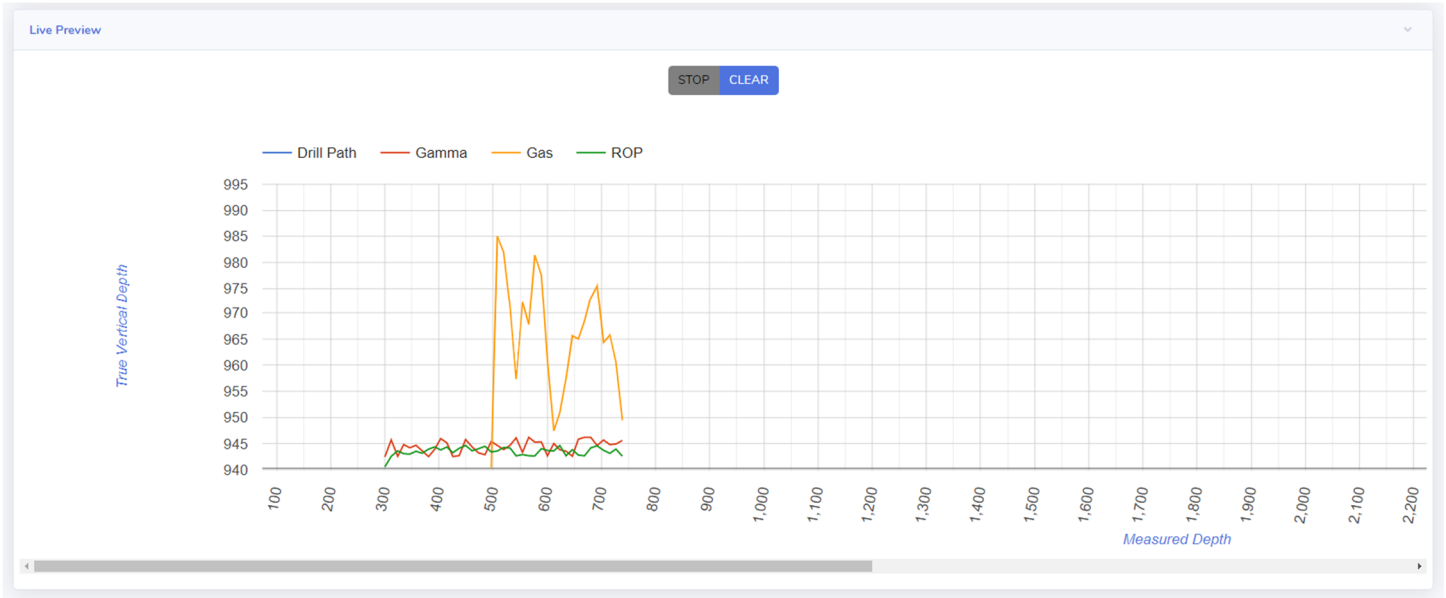


Figure 6 – Dashboard Page, Live Graph

3.2. Architecture

Refer to **Appendix A** for an architecture diagram that illustrates the overall design of the back-end components of the web app.

3.3. Functions

3.3.1. Serve Website/Authentication

The following diagram illustrates the process of serving websites and authentication from Firebase hosting

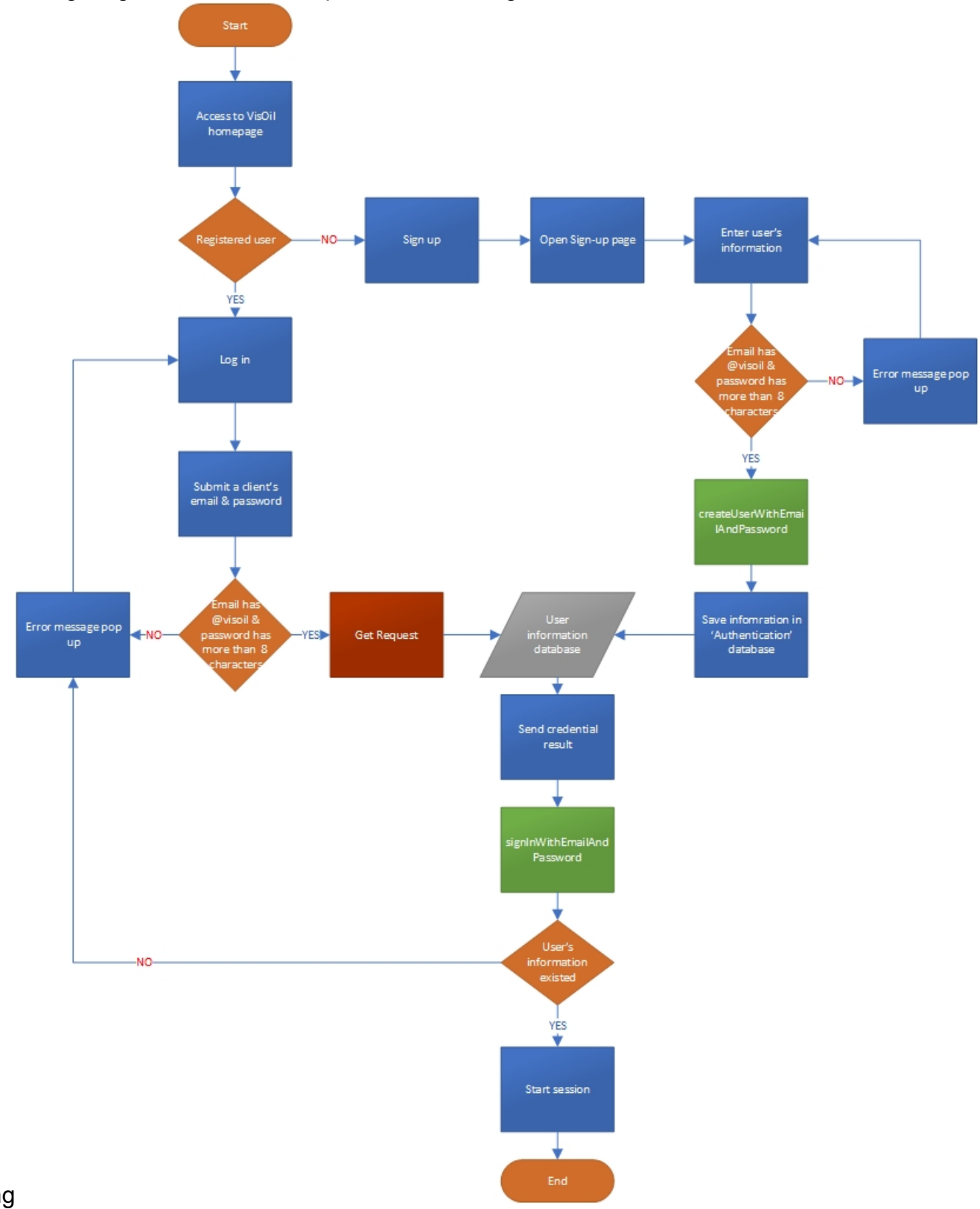


Figure 7 Visoil Web Hosting

3.3.2. Wellsite Selection

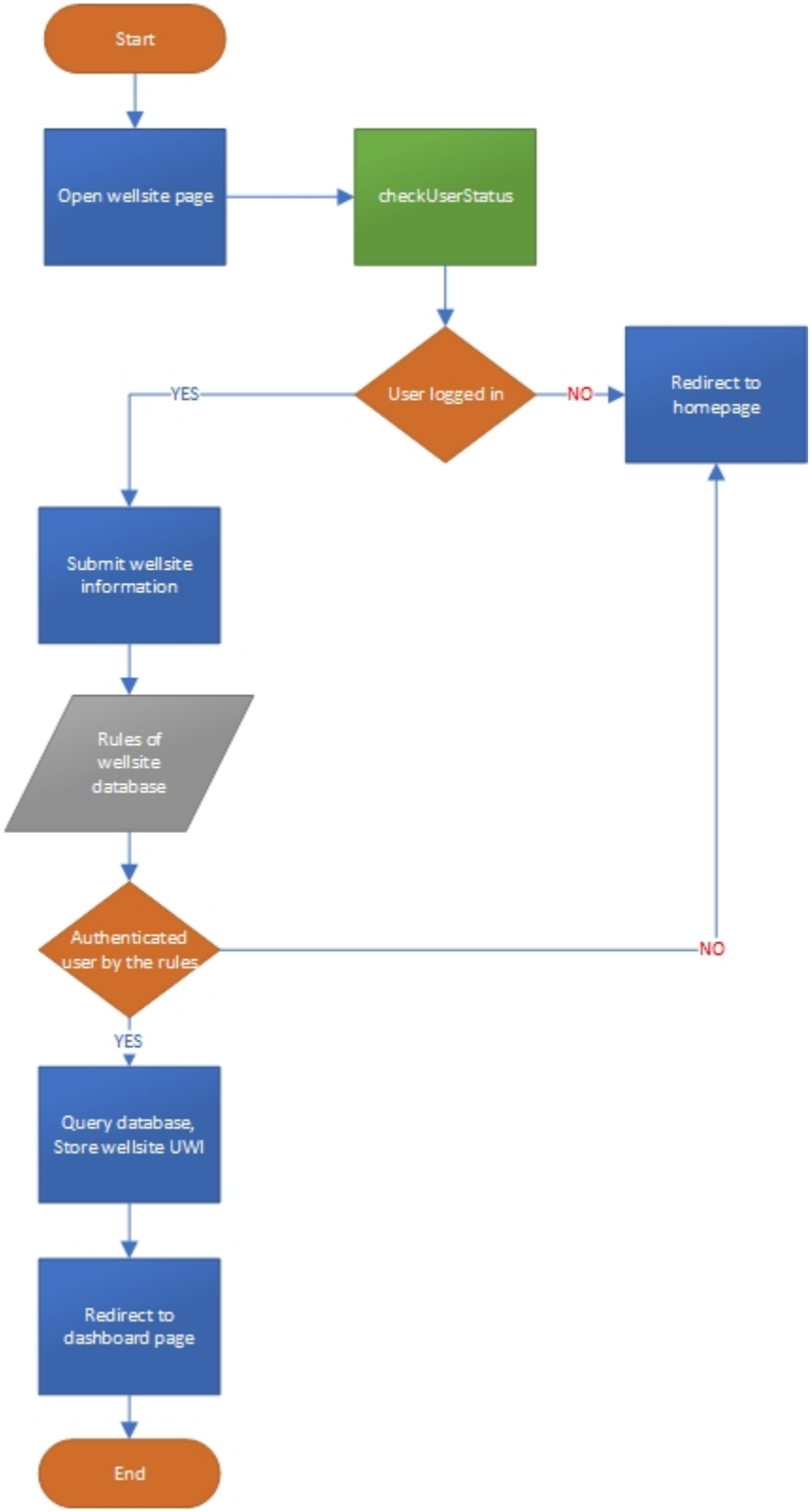


Figure 8 Visoil Wellsite Selection

3.3.3. Render Graph

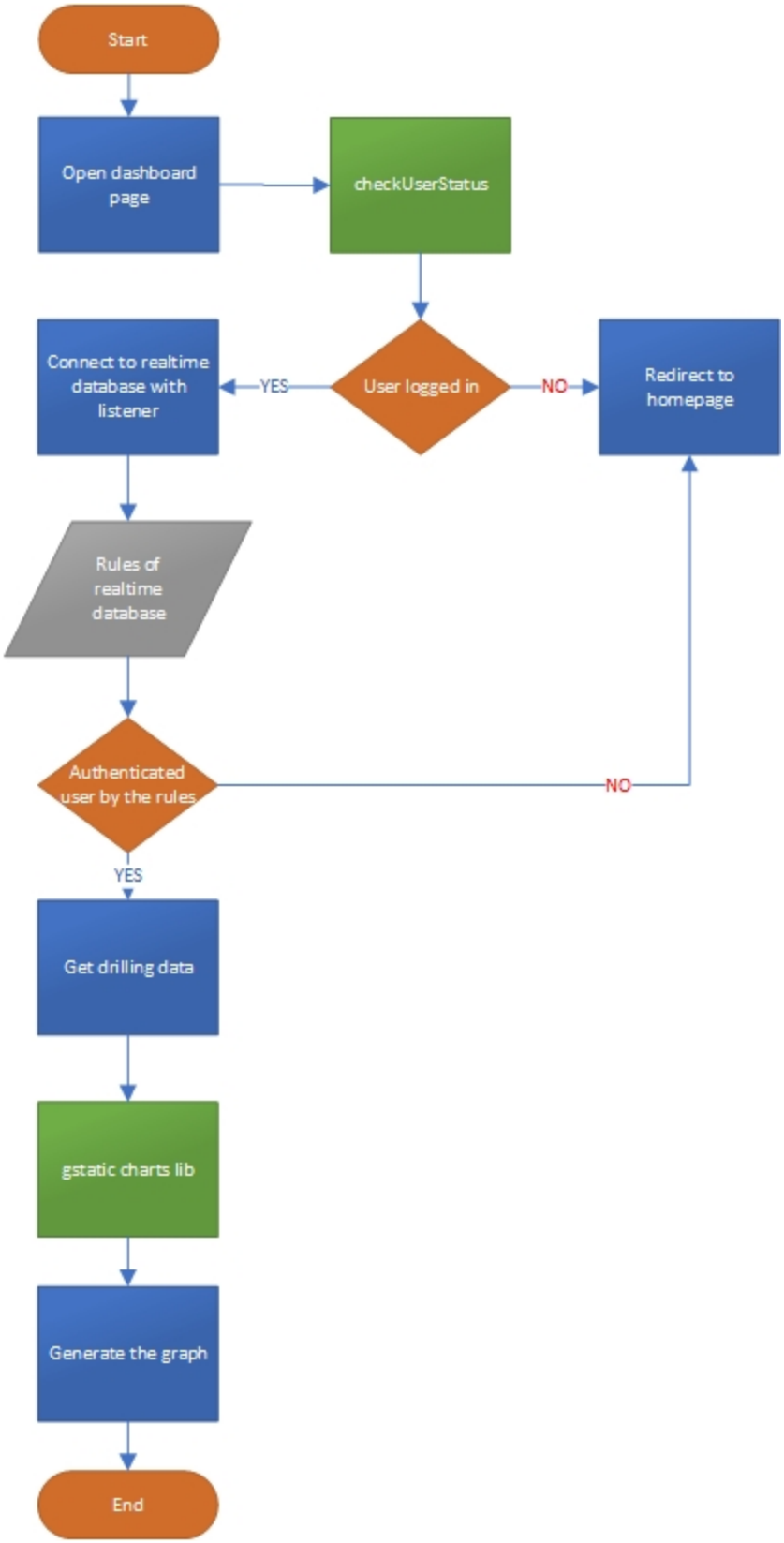


Figure 9 Visoil Graph

3.3.4. Real-Time Update

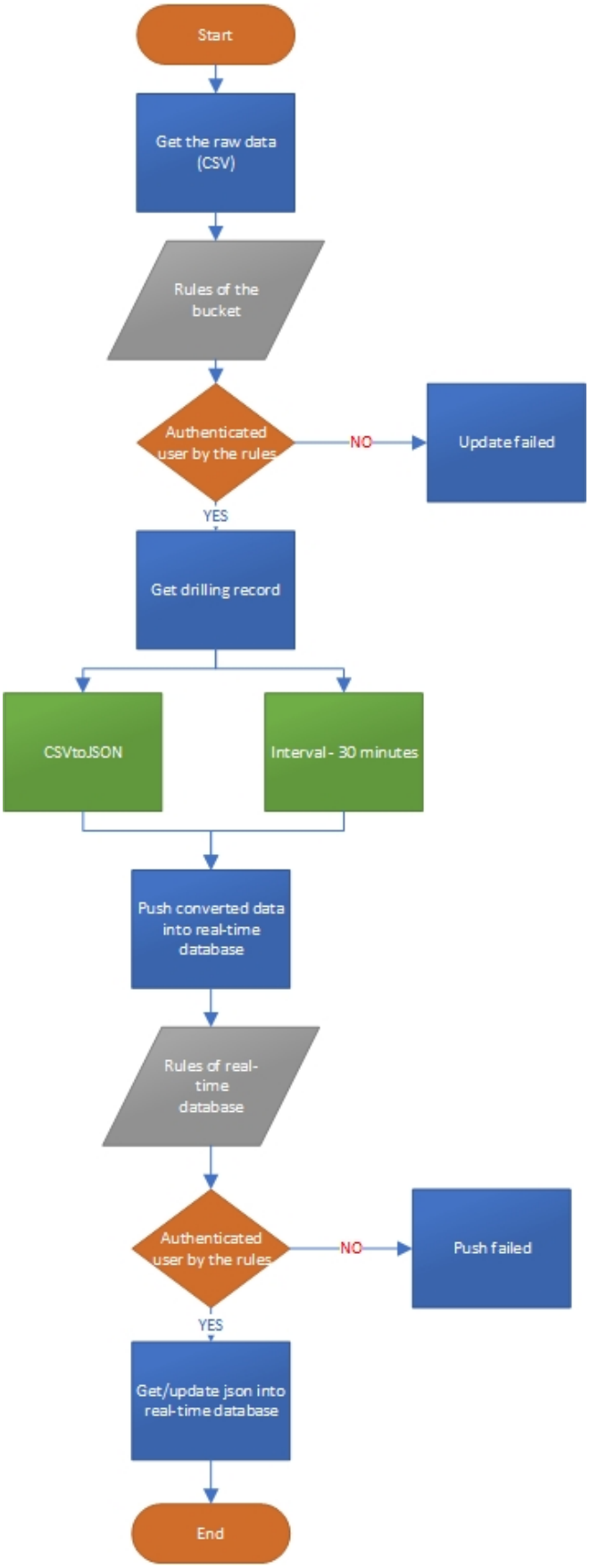


Figure 10 Visoil Real Time Function

4. Implementation

	Projected	Actual
Hardware	2 x HPE Proliant DL385 G7 2U rack server, 2 x 15-inch LCD monitor, 2 x USB keyboard, 2 x USB mouse, 6 x RJ-45 cable	3 x Toshiba Tecra C50-e Laptop
Software	MySQL, HTML, CSS, PHP , JavaScript, Windows Server 2016, Windows 10 Professional, CentOS Linux 7 1810, Kali Linux, MS Exchange Server 2016, Chrome browser, MS Excel 2019, pfSense, .NET Framework, Apache httpd, Visual Studio Code, Python, OpenVPN	HTML, CSS, JavaScript, Windows 10 Professional, Chrome browser, MS Excel 2019, Visual Studio Code, Firebase
Network	Networks are composed of three different sites/VLANs, and the sites can communicate each other via OpenVPN. A router between clients' machines and a web server can redirect network traffic from the Internet outside the LAN environment.	All network access are redirected via Firebase platform for administrators and clients both.
Security	IDS/Snort, OS hardening, SSL/TLS, VPN, Anti-virus application.	SSL/TLS, Firebase rule scripts.
Services	Web service, backup/restore service, DNS/DHCP services, Mail service.	Cloud Web App Hosting, Mail service, Cloud Database, Cloud Storage.

Due to the global pandemic (COVID-19) and the country-wide lockdown that followed soon after, our work was interrupted but didn't stop. Instead we researched and studied different solutions that were available to us at that time and decided that the most suitable alternative that also ensured the opportunity for our project to have all the features that we originally planned and agreed with all stakeholders was to change our implementation strategy fundamentally by moving to the cloud, which request got approved (see Appendix B *Change Request*).

4.1. Component Descriptions

4.1.1. Hardware

HARDWARE - Projected			
Item	**Qty.**	**Cost/Item**	**Subtotal**
HPE Proliant DL385 G7 (Server A)	1	$2,350.00	$2,350.00
HPE Proliant DL385 G7 (Server B)	1	$2,600.00	$2,600.00
LG 15-inch LCD Monitor	2	$120.00	$240.00
USB Keyboard/Mouse Combo	2	$30.00	$60.00
RJ45/Ethernet Cable	6	$10.99	$65.94
		TOTAL	**$5,315.94**

HARDWARE - Actual			
Item	**Qty.**	**Cost/Item**	**Subtotal**
Toshiba Tecra C50-e Laptop	3	$1,164.99	$3,494.97
		TOTAL	**$3,494.97**

Original hardware plan included necessary equipment for on-site hosting of the application but, due to the lockdown events and our decision to move our production environment to cloud-based hosting our list of hardware components shrank down to just bare minimum equipment – our laptops. Along with the equipment, our cost for hardware came down to just $3,494.97 managing to reduce it by 34.25% or $1,820.97 off of the original cost.

4.1.2. Software

Software - Projected				
Item	**Type/Edition**	**Qty.**	**Cost/Item**	**Subtotal**
Windows Server 2016	Standard	2	$399.99	$399.99
Windows 10 Professional	Standard	1	$186.90	$186.90
VMware ESXi	Enterprise	1	$4,350.00	$4,350.00
Exchange Server 2016 CU	Standard	1	$449.00	$449.00
Microsoft Excel 2019	Open	1	$231.00	$231.00
Chrome Browser	Open	1	Free	Free
			TOTAL	**$6,016.88**

Software – Actual				
Item	**Type/Edition**	**Qty.**	**Cost/Item**	**Subtotal**
Windows 10 Professional	Standard	3	$186.90	$560.70
Firebase	Spark Plan	1	$0.00	$0.00
Microsoft Excel 2019	Open	3	$231.00	$693.00
Visual Studio Code	Open	3	Free	Free
Git	Open	3	Free	Free
GitHub	Basic	3	Free	Free
			TOTAL	**$1,253.70**

Oroshi IT's development environment setup in addition to our operating system consisted of the cloud platform Firebase Spark, code editor VS Code; since we were working remotely, to track changes in

the source code Oroshi IT used Git as a distributed version-control system and GitHub to host our source code. This was a perfect setup for that particular situation which ensured that all members of the development team have access to the latest version of the project application and can successfully collaborate on it. In addition to that, the actual cost of the software was largely reduced by 79.16% or $4,763.18 compared to the original projected cost.

4.1.3. Networking

General information of Firebase's data like unauthenticated data or user information is transferred via HTTP. Large amount of data also is used in database and storage of Firebase to save times when the data is encrypted. For this project, Oroshi IT uses only HTTPS because the raw data and the number of users is low compared to a big company, and the graph data should be secured because the data is for oil and gas industry. Therefore, other competitors cannot access to the sensitive raw data or manipulate its database.

4.1.4. Security

All security protocols pass through **SSL**, authentication functions of Firebase. A client can register his/her account on the VisOil website via 'Create Account' page and enter their email address with 'visoil.ca' domain and password. Inputs are passed to database storage in the Firebase authentication section after 'signInWithEmailAndPassword' function gets and validates the information. When validations of the information are verified, the client's information is stored as '**UUID'** with applying hash-algorithm in the authentication database. A verification email to confirm the registration is sent by the 'Email address verification' template on the authentication function as well.

A client is required to enter his/her authentication credentials to access graphical information of well-sites when they are on the homepage of the VisOil web application. If the client is verified via their UUID in the stage, he/she has controls to access the drilling information by the rules of Firebase database and storage security.

Firebase **Real-time database** and storage are implemented by JSON, so the Firebase platform is secure for **SQL** database injection attacks. Firebase uses **APIs** with the provided **SDK** and sends values which are managed by the APIs. All entry of Firebase is restricted by the rules for the database and storage [2].

The rules of this project are limited to non-authenticated users. All authenticated users who are passed through the rules, they can access the well-site database to read and write with static and dynamic graphs both. The storage rule is configured to give only authenticated users permissions to read and write for all paths of the bucket. If a suspicious user tries to access the database and files in the storage, the Firebase refuses to give the user permissions. It leaves a log on 'Monitor rules' for security maintenance.

4.1.5. Services

The following services are offered by the Visoil Drilling Data Web App: Cloud web app hosting, email integration, real-time cloud database services, and cloud storage.

Being hosted from Firebase gives the project a progressive edge on traditional web hosting methods; Visoil is backed by the services offered on the Firebase platform such as application security, fast loading times, and network resilience [3], making it a progressive web application. All requests to and from the database, server, and storage are done over HTTPS, making the data secure from

interception. Overall the delivery of Visoil from a cloud platform makes it a highly effective web application that is easy to use, secure and fast.

Visoil seamlessly integrates with existing email services by providing an effective way of authenticating users securely with an email link. The user's information and password are never passed in a redirect URL, bur rather stored to local storage to eliminate the possibility of session injections [4]. Using email verification means that the user's password is never stored in a database that can be easily compromised, and removes a vector of attack making it more secure. Lastly, the application allows for blocking or allowing user sign up by domain, which gives the owner of the application the ability to restrict the usage to a single company.

The data for the drilling data graphs is stored in a NoSQL Firebase Realtime Database, which offers the ability to sync data from the database to the client in near real-time speeds [5], making it deal for use as a live drilling monitor. Being hosted from Firebase, the data is highly available, and is stored as JSON objects which are retrieved quickly, giving a smooth user experience.

Finally, Visoil makes use of Firebase Cloud Storage, which can be easily accessed with by server-side and client-side code. This means that user-generated data, such as reports or graph prints can be quickly sent to the cloud storage for safekeeping, and can be retrieved on-demand as needed.

All these services that the Visoil Drilling Data Web App is built on ensure that it is a progressive web application that will reduce customer costs, and deliver a user experience that would not be possible from a locally hosted webserver.

4.2. Challenges

There were several challenges faced during all phases of the project, which came from both external and internal origins.

The greatest challenge that the project faced was a global pandemic that struck just before project execution. This disrupted production, diminished resources, and forced the team to work remotely. Drastic changes to the implementation plan for the application needed to be made quickly, while keeping the customer's needs at the forefront of every decision. Since there was no longer a means to develop and host the application from a physical server, the team decided to move to the cloud, which had a cascading effect on the rest of the project. Each team member needed to quickly adapt to new interfaces, platforms, frameworks, and most importantly, collaboration and management methods.

The project management structure changed from a typical waterfall setup to a more agile delivery system, whereby features were developed and released successively, rather than all at once at completion. Team members adapted new technologies for collaboration, such as Git/GitHub, and worked towards continuous integration and delivery of the client's requirements.

Team members working on writing back-end code for the application had to learn on-the-fly how to deliver the application from Firebase, which meant understanding the platform, learning API calls to the Firebase services, and managing a whole range of new technologies that were not part of the original plan

Some other challenges that the team members faced were due to the nature of the drilling data graph, which consisted of about 75,000 data points in total. Firstly, this meant that the graph would

take up much of the user interface, and proved to be a difficult element to manage in a browser. The problem of the graph's size was solved by carefully implementing a minimal Bootstrap templated interface that make viewing the graph easy. Secondly, this meant that the data download from the database would be enormous, and would incur heavy costs; both time-wise and money-wise. The loading time for the graph proved to be extremely problematic, with delays on the client-side ranging up to 2 minutes or more. Additionally, since the team was working on a free-tier Spark Plan [6], they had to be careful not to go over data limits. Both these issues were solved by reducing the amount of data points by up to 99%, while maintaining a graph structure that closely resembled sample data given by the client.

In the end, the project team managed to deliver the client's original requirements for the product, despite a complete change in the delivery and implementation of the web app.

5. Results and Evaluation

5.1. Goals Achieved

Of the original requirements set out by the client, the team managed to overcome great obstacles to deliver a web application that met or exceeded the user's expectations as a proof of concept.

Functionally, each of the client's requirements were accomplished. The team successfully demonstrated that the drilling data samples given could be automatically formatted and displayed in a graph that is interactive and secure. Layers for the graph are able to be toggled on and off, and real-time simulation of data being delivered and formatted on-the-fly is viewable from the interface. Users are able to sign up and sign in with an email address, and the project owner is able to restrict user sign-ups to a single custom domain. Finally, the graph is able to be printed easily straight from the web interface, eliminating the need for downloading static versions of the graph. The application was built modularly, and while the features delivered were successful, integrating new features or functionality to the application would be easy for developers.

On the business side, the project accomplished its goals of completely eliminating the need for manual data entry and formatting, which greatly reduces costs and frees up time for other tasks. The graph is delivered in real-time, which also eliminates the need to manually update the graph at intervals throughout the day; instead, the user is able to log in and quickly view the progress of a live wellsite, and can immediately give feedback to the drilling operation team.

In summary, the Visoil project has delivered a web application that is able to save time and money for geologists and drilling teams alike, and is hosted on a platform that makes it easily scalable to meet any future needs that may arise.

5.2. Critical Tests & Results

Since the implementation of the project was focused on continuous delivery, the application was tested extensively to ensure each version delivered was a fully-functional application. The major releases of the application were:

1. Static graph
2. User Verification
3. Select Wellsite + Layer Toggle
4. Live Graph + Print to PDF

At the first stage, the graph was tested as a static display of the drilling data points. The test was to see if a browser could successfully connect to the Firebase Database, retrieve the data as JSON objects, parse the JSON into data tables, and format those tables into graphs. These tests were all successful.

In the second stage, user verification was successfully tested to see if users could be added to the web app and if access to application pages could be restricted to only authenticated users. Testing was done to try and circumvent the sign-in page, as well as viewing data while not authenticated.

Wellsite selection was successfully tested with the creation of several datasets in the Realtime database, and the user is able to choose from a dropdown that is populated by the Unique Wellsite Identifiers that represent each dataset. Layer toggling was tested though different combinations of toggles on the graph, and some issues were discovered where the layer that the user wanted to turn off also triggered other layers to not display. Due to time constraints and limitations to the functionality of the graph, this issue was not resolved.

Printing of the graph was tested, and extensive formatting options were explored to ensure that the graph was readable when exported to PDF.

Finally, a live display of dynamic data was successfully tested with the use of a function that set an interval to send the data to the Realtime database, simulating an operational drill. This data was successfully retrieved through a live ‘listener’ function that updates the graph each time data is added to the database.

6. Future Work

6.1. Non-Implemented Features

Although the customer’s original requirements for the web application were satisfied upon project completion, several additional value-added features were brought up during meetings with the client. These included extra ways to interact with the graph, as well as new and improved means of storing and loading the graph data. As a proof of concept, Visoil was greatly successful, but in a real-world application it’s functionality could be greatly extended to further increase the user experience.

A common need for geologists and operations teams is to share critical information while the drill is boring. One easy way to implement this is though a messaging feature that allows authenticated users to pass and retrieve notes to each other through the use of the real time database. It is also

common for geologists to add flags and comments directly to the graph, and while it is difficult to persist data in a browser, cloud storage and the printing feature could be implemented to store markups for the graph which could be easily retrieved and shared.

Finally, the problem of downloading and displaying the large amount of data required to render a full graph of all the datapoints could be implemented with the use of a client-side database that resides in the browser; IndexedDB. IndexedDB is a high-performance solution for client-side storage that has almost not data limit [7]. Through this database, the entire drilling dataset could be stored and retrieved without incurring download costs, while increasing performance.

Although these features were not implemented as part of the project, the design and structure of the Visoil web app has made it easily scalable for them as future work, with minimal effort needed on the developer's side, and fast delivery for clients who would like to see them implemented.

7. Conclusions

The Visoil Drilling Data Graphical Display web application is a solution for geologists who monitor directional drilling operations. The project aimed to eliminate manual data entry and graph creation, increase data security and availability, while reducing response time through live display of drilling data. The project successfully implemented a proof of concept for all these requirements through the use of a cloud hosted platform, and all features were delivered on-time and under budget. The project team was successful in quickly adapting to major external changes, and was able to quickly develop the skills and knowledge needed to ensure that the client's needs were met.

The final delivery of the project is a web application that displays drilling data as a real-time graph that is interactive and secure, and its features are easily scalable for future work. Although some issues remain with the layer toggling, the proof that it works as a concept has been achieved, and all other tests were successfully passed.

8. Reflections

8.1. Transferable Skills/Knowledge Gained

This project had three phases: design, development and delivery. Through-out each phase of this project we gained invaluable knowledge and skills. Starting with analyzing situations and data, planning budgeting setting goals and scheduling tasks and activities, keeping records, coordinating activities, researching technical and non-technical information, delegating tasks or responsibilities, speaking in public, supervising others' work, troubleshooting issues or problems, testing for quality, documenting processes, updating technical documents, handling complaints, motivating others etc.

8.2. Lessons Learned

Addressing the problem of manual data entry in the workplace is a common need for many companies. While this project's approach to eliminating tedious and time-consuming tasks was one way of accomplishing that goal, there are several possible solutions that could have been applied. This project succeeded in proving that a web application could solve this problem effectively, but where it fell short was in the area of added-value. Manipulating and editing live content from a browser, while persisting changes over several sessions is extremely difficult and unnecessarily complicated. A desktop application would be better suited for customizations and data manipulation, whereas the web application is good for quick and reliable status updates, as well as the possibility of real-time collaboration and information sharing. The decision to implement this project as a web application was a critical first step in determining the direction of the entire project, and the project team believes that it was the correct decision. Given the time and budget constraints that were included with the project, this decision proved to be instrumental in the project's success, especially when faced with sudden changes that required the team to deliver the end product while having to work and collaborate remotely.

9. Glossary

9.1. Acronyms

CSV	Comma-Separated Values
CSS	Cascading Style Sheets
ESXi	Elastic Sky X integrated
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HTML	Hypertext Markup Language
SAIT	Southern Alberta Institute of Technology
SSL	Secure Sockets Layer
JSON	JavaScript Object Notation
JS	JavaScript
UWI	Unique Well Identifier
MS Teams	Microsoft Teams
VPN	Virtual Private Network
UUID	Universally Unique Identifier
SSH	Secure Shell
SQL	Structured Query Language
API	Application Programming Interface
SDK	Software Development Kit

9.2. Terms

Bootstrap	A framework for quickly designing and customizing webpage styles and user interface features.
Firebase	Mobile and web application development platform
Firebase Spark	Firebase pricing plan
Firestore	Cloud-hosted database of Firebase
Real-time Database	Advanced cloud-hosted database of Firebase
IndexedDB	A client-side API for large storage requirements in the browser.
OAuth	Industry-standard protocol for authorization
Oroshi IT	Developer
Git	Distributed version-control system
GitHub	Software development cloud-based hosting and version control using Git
VS Code	Free source-code editor
VisOil	Project Name
Google Cloud Shell	Browser-based command line interface shell for Google Cloud Service
HASH-Algorithm	Algorithm to generate random values with fixed size
gsutil	Python-based Google Cloud management tool

10. References

[1] Google Developers, "Firebase helps mobile and web app teams succeed," [Online]. Available: https://firebase.google.com/. [Accessed 07 07 2020].

[2] "Firebase Authentication," Google, [Online]. Available: https://firebase.google.com/docs/auth. [Accessed 17 July 2020].

[3] Google Developers, "Use Firebase in Progressive Web Apps," Google , [Online]. Available: https://firebase.google.com/docs/projects/pwa. [Accessed 16 07 2020].

[4] Google Developers, "Authenticate with Firebase Using Email Link in JavaScript," Google, [Online]. Available: https://firebase.google.com/docs/auth/web/email-link-auth. [Accessed 16 07 2020].

[5] Google Developers, "Firebase Realtime Database," [Online]. Available: https://firebase.google.com/docs/database. [Accessed 16 07 2020].

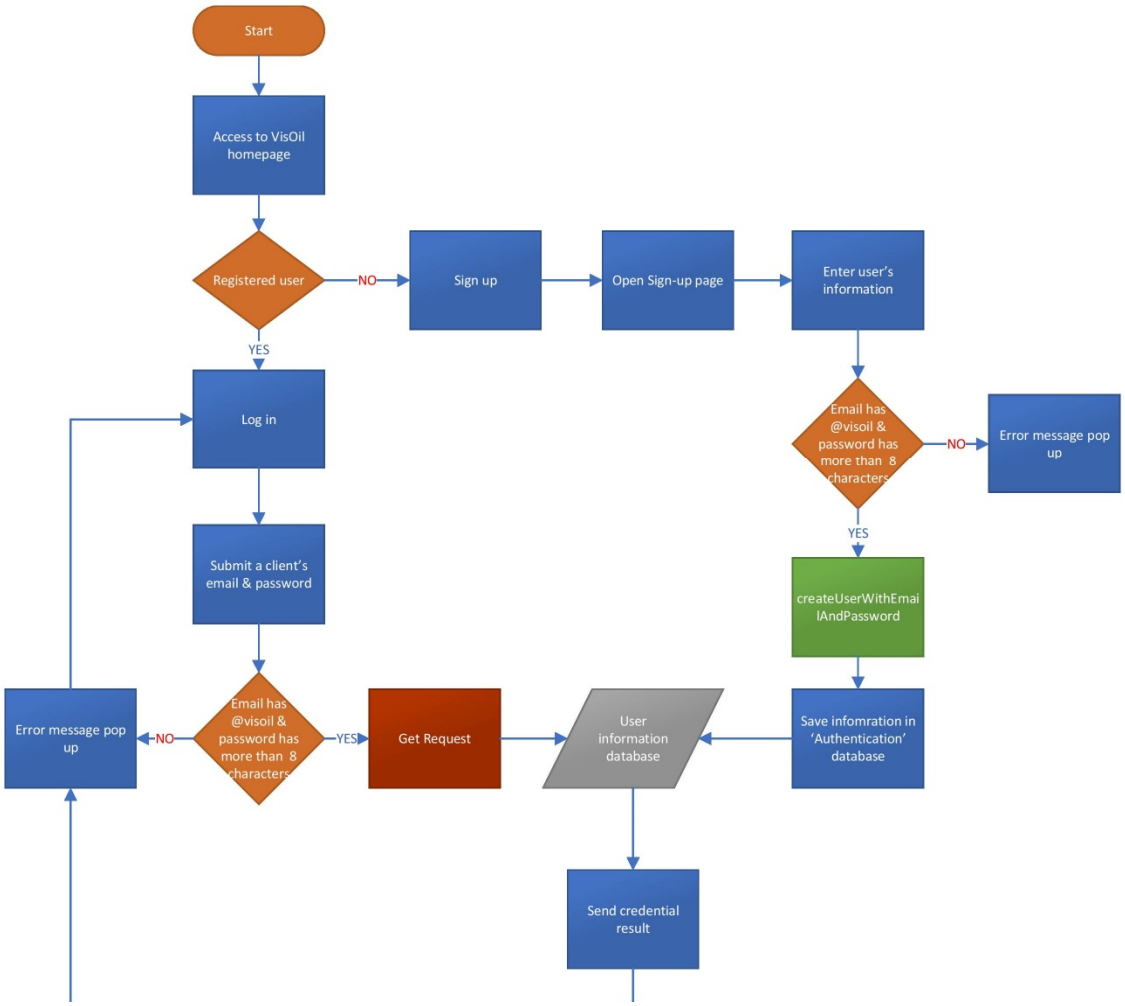
[6] Google Developers, "Pricing plans," Google, [Online]. Available: https://firebase.google.com/pricing. [Accessed 16 07 2020].

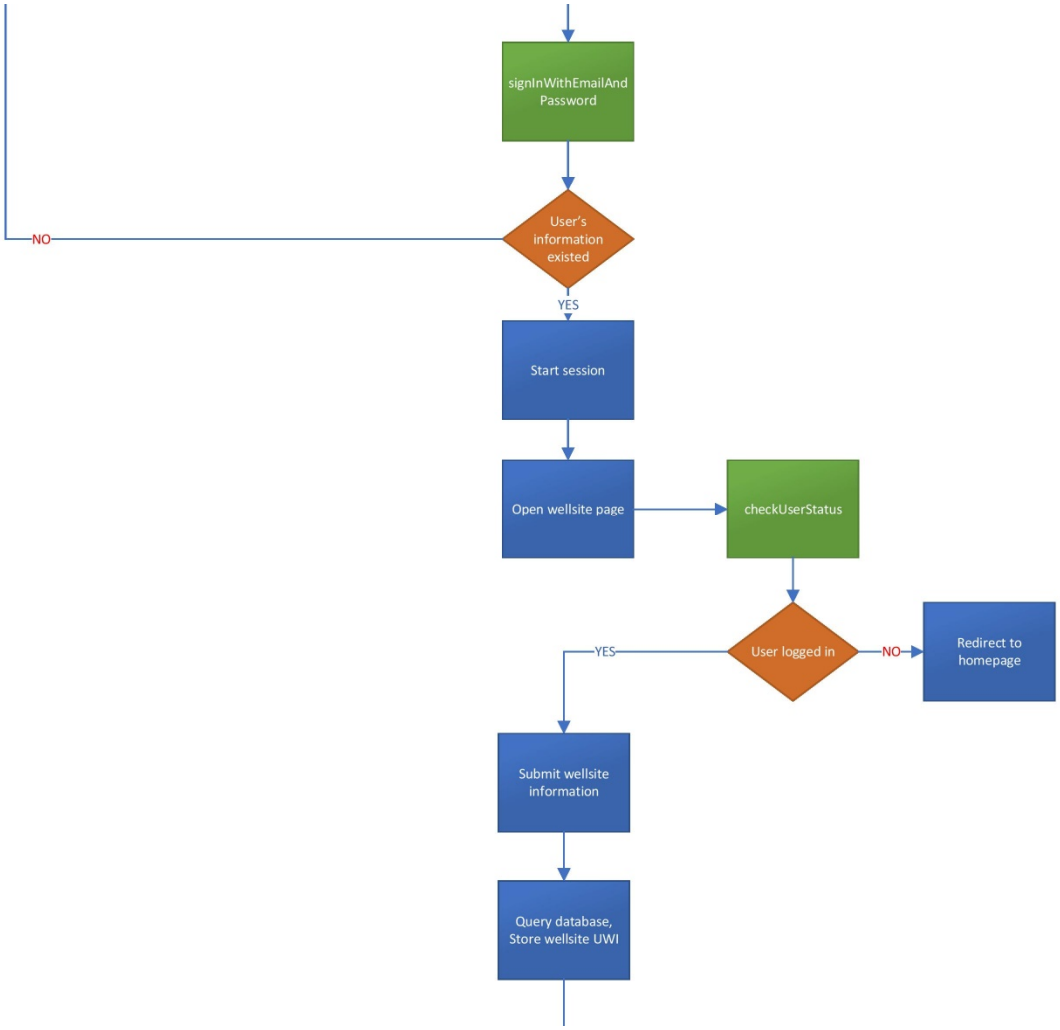
[7] MDN Web Docs, "IndexedDB API," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. [Accessed 16 07 2020].

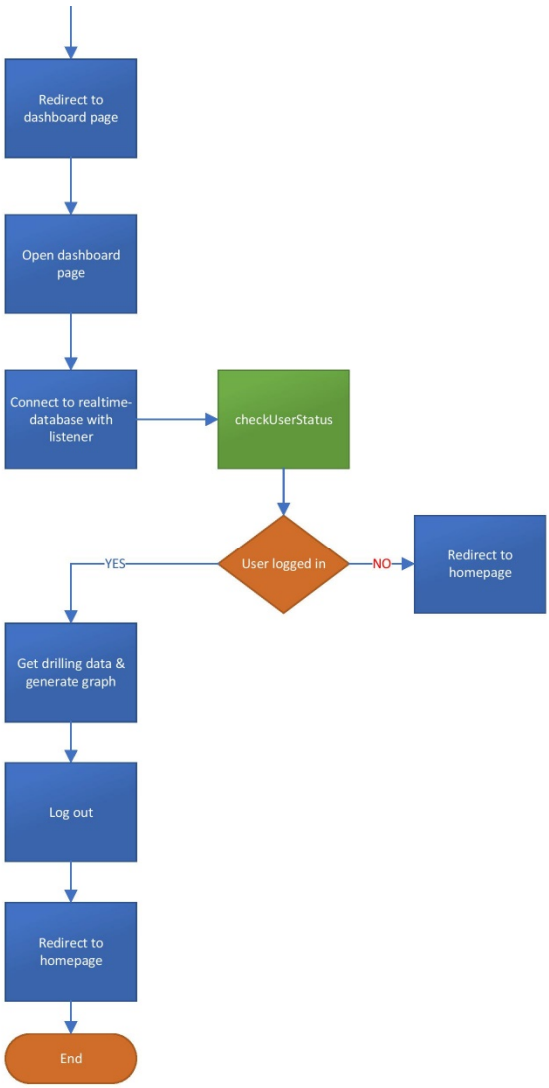
[8] Software Freedom Conservancy, "Git," [Online]. Available: https://git-scm.com/. [Accessed 14 07 2020].

[9] Wikipedia, "VMware ESXi," [Online]. Available: https://en.wikipedia.org/wiki/VMware_ESXi. [Accessed 16 07 2020].

11. Appendix A – *Application Architecture*




signInWithEmailAnd Password
User's information existed
NO
YES
Start session
Open wellsite page
checkUserStatus
User logged in
YES
NO
Redirect to homepage
Submit wellsite information
Query database, Store wellsite UWI


Redirect to dashboard page
Open dashboard page
Connect to realtime-database with listener
checkUserStatus
User logged in
YES
NO
Redirect to homepage
Get drilling data & generate graph
Log out
Redirect to homepage
End

12. Appendix B - *Request for Change #001*

Request for Change #001	
Project: VisOil – Drilling Data Graphical Display	**Date:** 2020-05-19
Change Requestor: Thomas Llewellyn	**Change No:** 001
Change Category (Check all that apply): ☐Schedule ☒Cost ☒Scope ☒Requirements/Deliverables ☒Testing/Quality ☐Resources	
What Kind of Change? (Check all that apply): ☐Corrective Action ☐Preventative Action ☐Defect Repair ☐Updates ☒Other	
Describe the Change Being Requested: Move application hosting from on-site hardware to free-tier cloud hosting solution.	
Describe the Reason for the Change: Pandemic; In-house hosting facilities closed, no access to hardware.	
Describe all Alternatives Considered: - Changing scope to no longer require web access - Hosting from private network/laptop - Paying for 3rd party vendor	
Describe any Technical Changes Required to Implement this Change: Will require implementation of Cloud resources: storage, database, hosting, and application. Cloud hosting also requires account & subscription. Infrastructure requirements such as hardware, network, security and services will now be provided by cloud provider, instead on in-house. All configurations will be done in a virtual environment instead of on physical hardware. Application features and development will remain functionally the same, but additional integrations with cloud provider will need to be made.	
Describe Risks to be Considered for this Change: Lack of experience with cloud hosting Unforeseen costs from hosting in cloud, should application exceed free tier allowance.	
Estimate Resources and Costs Needed to Implement this Change: Cloud Hosting Account/Subscription (e.g. Google Cloud Services, AWS, Azure, etc.) Project team will require time to learn how to implement cloud hosting.	
Describe the Implications to Quality: The final application will maintain it's original functionality, but with a predicted increase in availability, increasing the quality of the deliverable.	
Disposition: ☐Approve ☐ Reject ☐ Defer	
Justification of Approval, Rejection, or Deferral:	