ReportAid

by

Stephen Lucas

John Parrick

An Interactive Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

May 13, 2020

APPROVED:

Dr. Therese Smith

Abstract

The goal of this project was to fix the various UI issues with the ReportAid GitHub project. Our initial focus was fixing the UI menu so it would resize proportionally to the window’s current size. While working on this project, we had discovered several issues that impede our progress. Most of these issues related to the installation of ReportAid, though there were a few issues with the window resizing.

Table of Contents

Table of Figures

Introduction

The goal of this IQP is to contribute to an ongoing non-profit software project. After searching through Github, we found a few projects that we were interested in and felt that we had sufficient skills to contribute. We chose ReportAid because it clearly identified the tasks to be worked on. We were also interested in the project because it would enable humanitarian groups to use an improved method when doing aid reports. Three years ago, at the UN World Humanitarian Summit in Istanbul, an agreement was reached that a new report for aid methods would be essential. One possible method to improve aid reports is through blockchains. Blockchain could allow for a more efficient and secure way to manage the finances of humanitarian aid. The project proprietor and primary programmer is Steve Huckle, a Ph.D. student at the University of Sussex. He also goes by Glowkeeper.

Our primary objective over most of the IQP period, October 2019 to May 2020, has been to fix the various UI issues. The main priority of the project was an issue with resizing the menu bar. The content of the menu bar did not change size when the width of the ReportAid window changed. We were able to uncover many bugs during the installation and setup process. These complications impeded progress and prevented the program from compiling or working. By the end of the IQP period, we had not found a plausible solution for our initial goal of resizing the menu items. The Humanitarian toolbox IQP has given us experience in jumping into unfamiliar projects, learning unfamiliar program languages, difficulties of installation of various platforms, gaining project group skills, and gaining real-life work experience through communicating with the team.

Background: ReportAid

Blockchain is commonly used in banking, investment, and cryptocurrency. It is used to keep track of financial transactions between participants. Each block stores transaction information and chains them together in chronological order. This process makes it easy to track the movement of money while making it difficult for a single block to be doctored. "*This is the repository of ReportAid – a tool for reporting on humanitarian aid. ReportAid is a blockchain implementation of IATI* (International Aid Transparency Initiative) *Standard*." $_{3\text{(Huckle)}}$ The purpose of this program is to use blockchains to deliver trustworthy aid reports by making them transparent, to make sure that the funds are actually going to the qualified people in a timely manner. Blockchains are created in chronological close-knit stacks, and it is impractical to falsify transactions because of the excess computation required to change any information. The project was first drafted by another person, but they ended up pursuing a different purpose for the program. Steve Huckle wanted to continue the original concept and prove that it can be done. A significant focus that came out of the 2016 UN World Humanitarian Summit in Istanbul was referred to as "Grand Bargain." The precedence over finding a transactional tracking method that fits the "3 Ts"- traceability, totality, and timeliness. Potentially, blockchains could be used, but at the moment, there is limited research on its effectiveness and may require a costly technological overhaul to be implemented everywhere.

Background: Humanitarian Free and Open Source Software

Humanitarian Free and Open Source Software (HFOSS) is a type of open-source software intended to be used by non-profit organizations worldwide to help people in need. Higher education institutions have been adding HFOSS projects into their STEM programs. The benefits of experience working on a project are numerous, including; "improved student learning, increased motivation to study computing, attracting women to computing and increased appreciation of the societal impact of computing." [4] They also gain experience about doing a professional project. Not only do they learn about group coordination and organization skills, but they also experience more of exploring Free and Open Source Software (FOSS) communities like Github, fleshing out their programs with open community imports and learning how to distribute the finished software. The participants contributing to humanitarian open source projects inherently are welcoming to women and underrepresented minorities, and tolerant to students' emerging skills. Students get to increase their skills in computer science, the opportunity to work with a professional team, and a tangible project to showcase to potential employers.

Description of Work

The main focus of the work, while not handling the necessary humanitarian finances, is to fix UI issues. Specifically, our focus was on fixing the issue related to the menu bar. The issue with the menu bar was that it did not properly adjust when the window was resized. Our goal was to make the menu bar fit inside a percentage of the screen no matter the size of the window. The project requires knowledge of Git, JSX (JavaScript XML), TypeScript, Material-UI, and the program's internal workings. This required researching programs on the web, and then learning how to use them. In order to facilitate testing the fixes to the menu problem, ReportAid needs to be downloaded to our computer in a runnable format. Changes and improvements need to be tested directly with ReportAid to verify that it is working. The goal is to produce a testable software improvement. Once completed and verified, we will provide the changes to the project manager. We currently have the code, but we are still working on getting it to run. At several points, we have run into incompatible errors and reported them to the Github repository issues page. After reporting the issues, there were delays where we waited for any responses and fixes. Our initial approach was to utilize the response delay which could be as long as one or two weeks by trying to reverse-engineer the ReportAid UI JSX Code Sandbox emulator. Our Success was limited in creating a menu because that part of the code had deep relations to other parts of the program, so it was impractical to import all the necessities into a sandbox. A significant breakthrough was discovering that ReportAid had incompatibility problems with newer versions of Node.js. It was quickly fixed after bringing it to Huckle's attention, and he immediately fixed it. Without this intervention, he would not have known that his program only worked on his computer. After finishing setups, we focused on learning React, Redux, and TypeScript to understand where the UI bug was coming from and ways to fix it. As we worked and made changes to the main program, all our changes were documented and sent to a separate version of the repository pull. As we discovered new errors in the program, we posted on the ReportAid issue page and sent a report on the issue of the project page on Github and explained the problem we had.

We have the code but did not have all the components to run it because it depended on setting up other programs and specific console commands, that were unfamiliar to us.

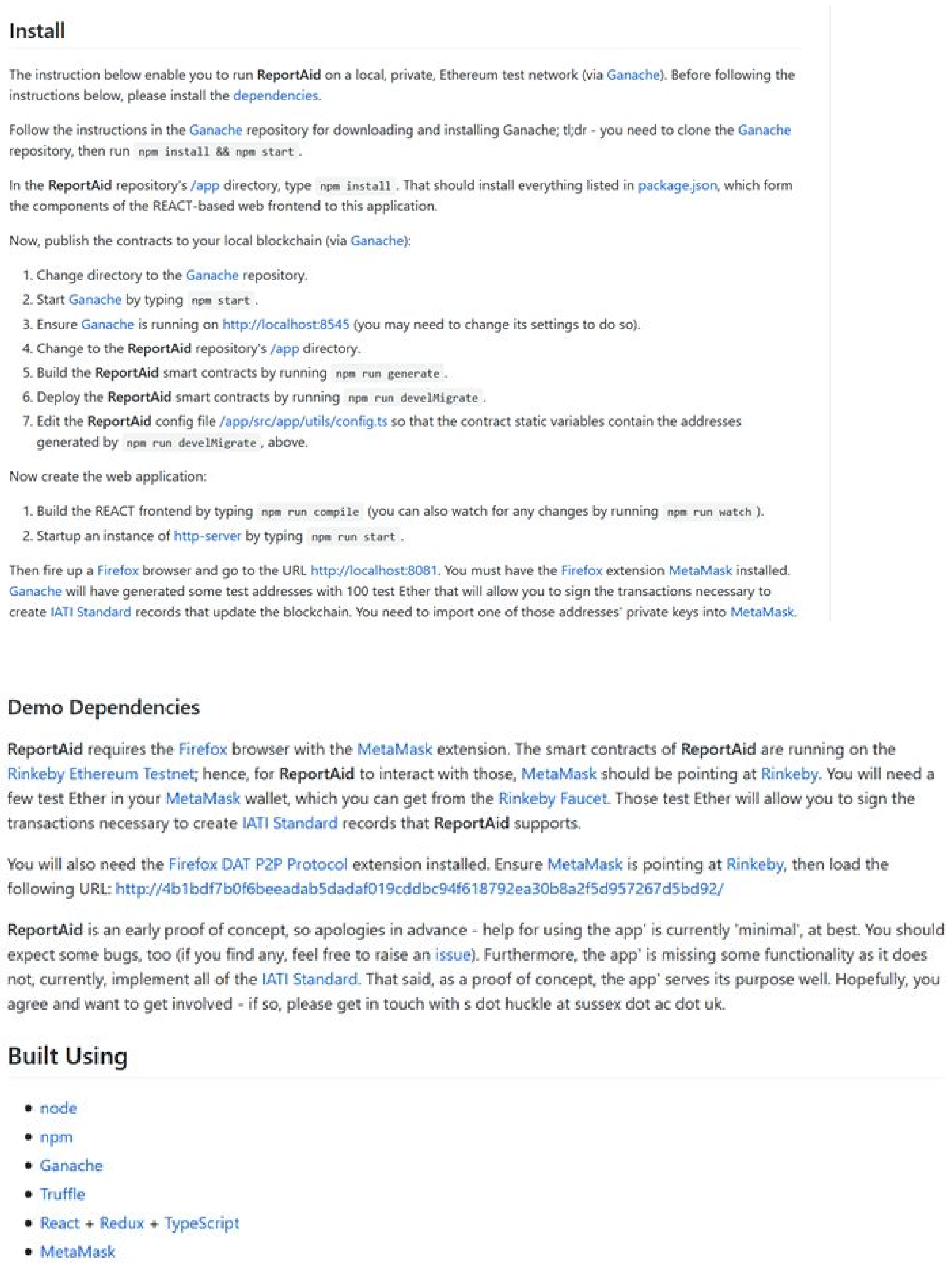
Installation of ReportAid

Installation of ReportAid required multiple steps,though they are clearly listed on the ReportAid Github repository page, the multiple downloads and requirements were confusing and time-consuming. In order for ReportAid to work, several dependencies had to be installed first. The first dependency is the MetaMask extension for Firefox. This extension would allow for the browser to easily interface with the blockchain application. MetaMask also had to be configured to point to the Rinkeby Faucet. The next dependency is the Firefox Dat P2P Protocol extension, which makes the dat://protocol available to be modified and accessed. Other dependencies required for ReportAid are node.js and npm. Node.js is a dependency for npm. Npm is a package installation service used to install the rest of the dependencies for ReportAid. Some of the last dependencies that must be manually downloaded and installed are Ganache and Truffle. Ganache is a blockchain tool used for developing blockchain applications like ReportAid. Truffle is a development environment for developing blockchain applications. The last dependency that must be manually downloaded and installed is http-server. This step is done by running the command "npm -I http-server". This is used to run the server that will host the ReportAid application.

After downloading Ganache, it will need to be installed by going to the Ganache directory and running "npm install && npm start". The command will install Ganache, along with any of Ganache's dependencies, and start Ganache. The next step requires going to ReportAid repository's/app directory and run "npm install", which will install any other dependencies for ReportAid.

Next, we will publish the contracts to your local blockchain using Ganache. To do this, first, we will start Ganache and ensure that it is running on http://localhost:8545. In the /app directory of ReportAid, we will run the following commands in sequence: "npm run generate", "npm run develMigrate". Lastly, we will modify the config file /app/src/app/utils/config.ts of ReportAid so that the contract static variables contain the addresses generated by "npm run develMigrate".

Finally, we will compile ReportAid by running "npm run compile" in the /app directory of ReportAid. Afterward, we start ReportAid by running "npm run start" In order to access ReportAid, all you have to do is open up a Firefox window and go to the url http://localhost:8083.



Install

The instruction below enable you to run **ReportAid** on a local, private, Ethereum test network (via Ganache). Before following the instructions below, please install the dependencies.

Follow the instructions in the Ganache repository for downloading and installing Ganache; tl;dr - you need to clone the Ganache repository, then run `npm install && npm start`.

In the **ReportAid** repository's /app directory, type `npm install`. That should install everything listed in package.json, which form the components of the REACT-based web frontend to this application.

Now, publish the contracts to your local blockchain (via Ganache):

1. Change directory to the Ganache repository.
2. Start Ganache by typing `npm start`.
3. Ensure Ganache is running on http://localhost:8545 (you may need to change its settings to do so).
4. Change to the **ReportAid** repository's /app directory.
5. Build the **ReportAid** smart contracts by running `npm run generate`.
6. Deploy the **ReportAid** smart contracts by running `npm run develMigrate`.
7. Edit the **ReportAid** config file /app/src/app/utils/config.ts so that the contract static variables contain the addresses generated by `npm run develMigrate`, above.

Now create the web application:

1. Build the REACT frontend by typing `npm run compile` (you can also watch for any changes by running `npm run watch`).
2. Startup an instance of http-server by typing `npm run start`.

Then fire up a Firefox browser and go to the URL http://localhost:8081. You must have the Firefox extension MetaMask installed. Ganache will have generated some test addresses with 100 test Ether that will allow you to sign the transactions necessary to create IATI Standard records that update the blockchain. You need to import one of those addresses' private keys into MetaMask.

Demo Dependencies

ReportAid requires the Firefox browser with the MetaMask extension. The smart contracts of **ReportAid** are running on the Rinkeby Ethereum Testnet; hence, for **ReportAid** to interact with those, MetaMask should be pointing at Rinkeby. You will need a few test Ether in your MetaMask wallet, which you can get from the Rinkeby Faucet. Those test Ether will allow you to sign the transactions necessary to create IATI Standard records that **ReportAid** supports.

You will also need the Firefox DAT P2P Protocol extension installed. Ensure MetaMask is pointing at Rinkeby, then load the following URL: http://4b1bdf7b0f6beeadab5dadaf019cddbc94f618792ea30b8a2f5d957267d5bd92/

ReportAid is an early proof of concept, so apologies in advance - help for using the app' is currently 'minimal', at best. You should expect some bugs, too (if you find any, feel free to raise an issue). Furthermore, the app' is missing some functionality as it does not, currently, implement all of the IATI Standard. That said, as a proof of concept, the app' serves its purpose well. Hopefully, you agree and want to get involved - if so, please get in touch with s dot huckle at sussex dot ac dot uk.

Built Using

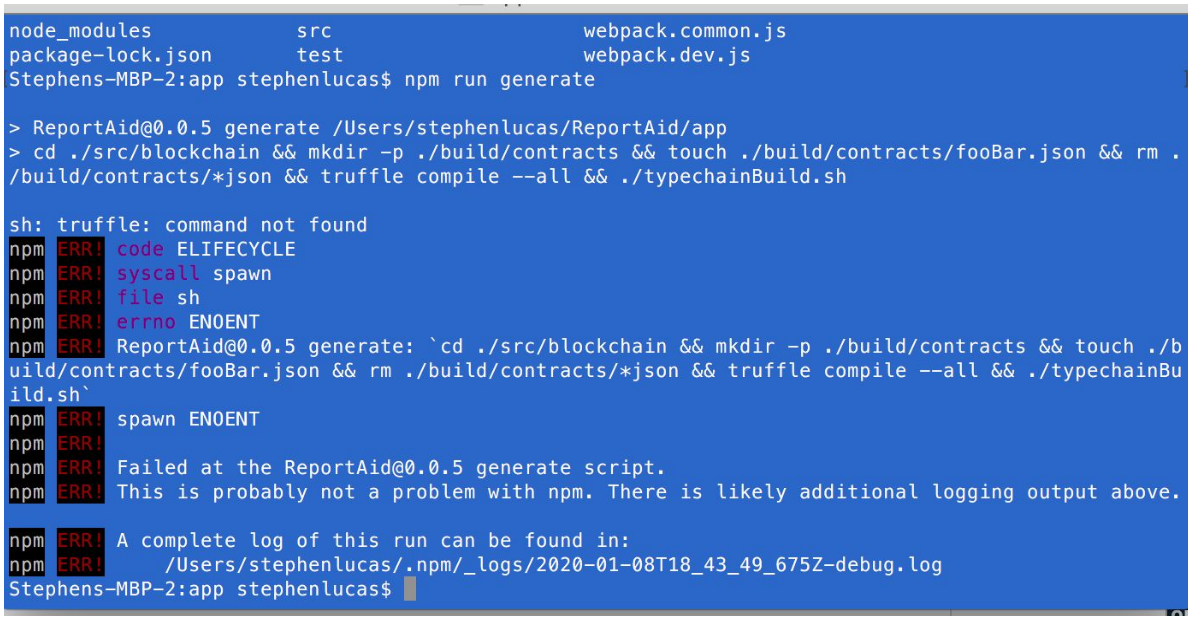
- node
- npm
- Ganache
- Truffle
- React + Redux + TypeScript
- MetaMask

Figure 1: Instructions and List of Dependencies on the Github Page.

Constraints/Challenges within the Project

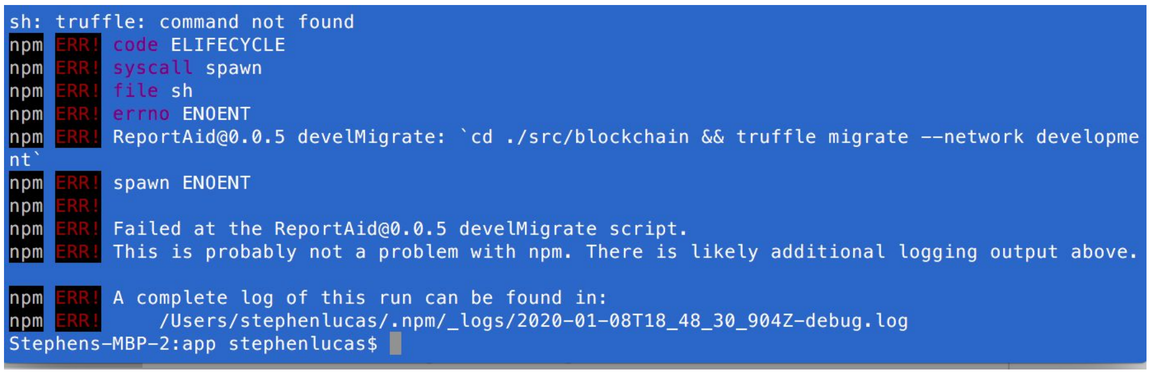
A significant hurdle throughout the project was installing ReportAid on our computers. We attempted to utilize simulators to get around installing the program so we could start identifying solutions and recommend improvements. Unfortunately, the complexity of the code requires that ReportAid needs to be installed directly on our computer for testing. The first obstacle with installation was that ReportAid's initial calibration was not compatible with the consumer version of node.js 11. It only worked on Steve Huckle's computer setup, but thankfully it was quickly fixed.

The next problem, discovered late November, was an error in a third-party program-Ganache had an update which resulted in missing an executable "start" script (see below) so we could not compile it. We were unable to resolve the error, thus halting our progress. We believe it has been there all along, but we had not noticed it. It is missing a runnable script, which required us to place an error report on the ganache Github page.



```
node_modules            src                     webpack.common.js
package-lock.json       test                    webpack.dev.js
Stephens-MBP-2:app stephenlucas$ npm run generate

> ReportAid@0.0.5 generate /Users/stephenlucas/ReportAid/app
> cd ./src/blockchain && mkdir -p ./build/contracts && touch ./build/contracts/fooBar.json && rm .
/build/contracts/*json && truffle compile --all && ./typechainBuild.sh

sh: truffle: command not found
npm ERR! code ELIFECYCLE
npm ERR! syscall spawn
npm ERR! file sh
npm ERR! errno ENOENT
npm ERR! ReportAid@0.0.5 generate: `cd ./src/blockchain && mkdir -p ./build/contracts && touch ./b
uild/contracts/fooBar.json && rm ./build/contracts/*json && truffle compile --all && ./typechainBu
ild.sh`
npm ERR! spawn ENOENT
npm ERR!
npm ERR! Failed at the ReportAid@0.0.5 generate script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/stephenlucas/.npm/_logs/2020-01-08T18_43_49_675Z-debug.log
Stephens-MBP-2:app stephenlucas$
```

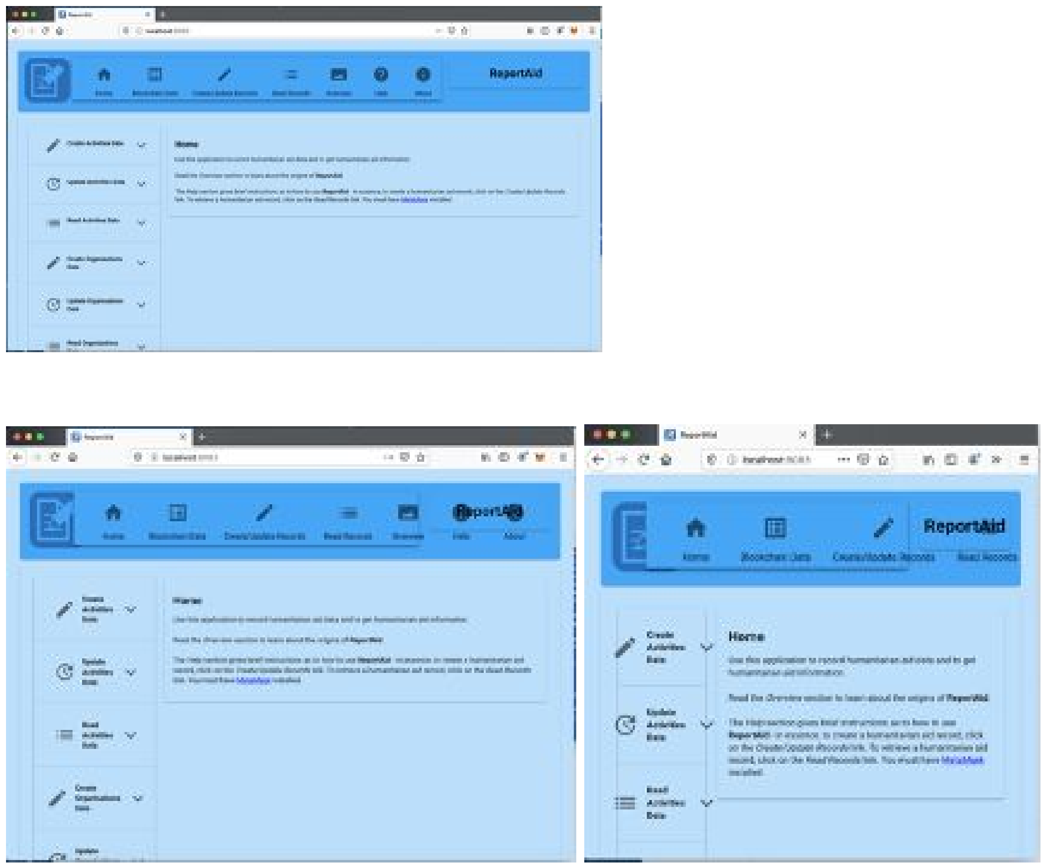
Figure 2: Error when Compiling Program..



```
sh: truffle: command not found
npm ERR! code ELIFECYCLE
npm ERR! syscall spawn
npm ERR! file sh
npm ERR! errno ENOENT
npm ERR! ReportAid@0.0.5 develMigrate: `cd ./src/blockchain && truffle migrate --network developme
nt`
npm ERR! spawn ENOENT
npm ERR!
npm ERR! Failed at the ReportAid@0.0.5 develMigrate script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/stephenlucas/.npm/_logs/2020-01-08T18_48_30_904Z-debug.log
Stephens-MBP-2:app stephenlucas$
```

Figure 3: Error messages from Ganache

We were not the only ones experiencing errors. There were multiple complaints posted on Github, so it eventually was resolved with the next update sometime early January. Now that Ganache was working, we moved on to build and deploy smart contracts but overlooked an explicit instruction that required that Report aid's app/ directory needed to be installed with npm, setting up the rest, and also how to start up the program casually. After that, we were partially successful in setting up and having access to a running version of ReportAid. Our Attempts to recompile the program with changes to the program failed, and then it became difficult again to reinstall ReportAid.



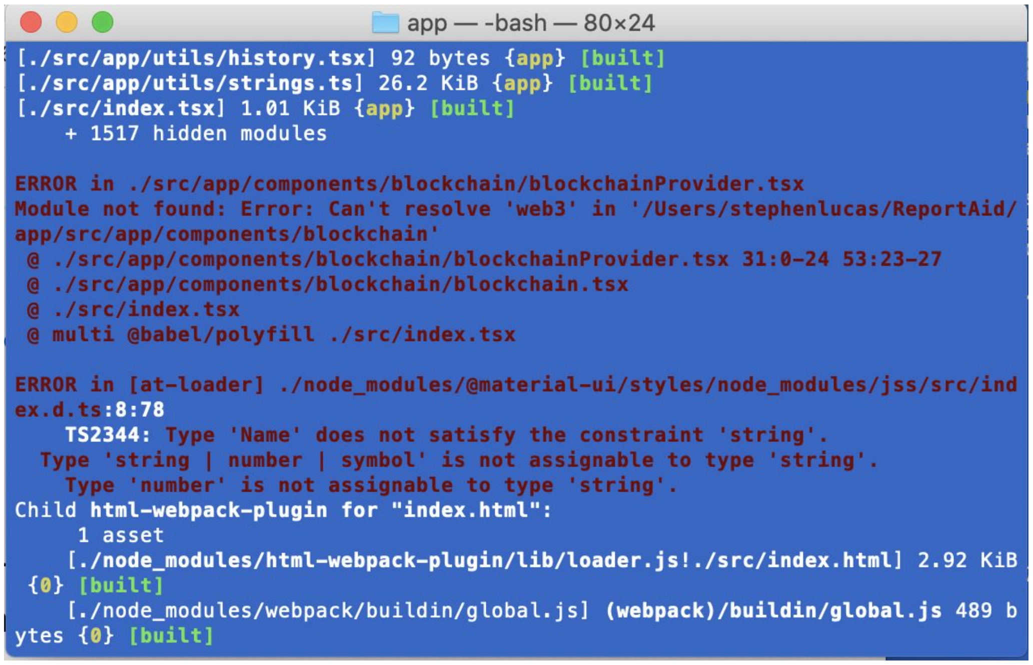
Figures 4: Example of ReportAid's Menu.

With it installed, we now needed to learn how the program controls the menu's UI and how to fix it best. The UI, in addition to being all shaded blue, the top menu bar contents do not fit when the window shrinks. We had ideas on how to apply changes to the program but unable to test our possible solutions despite the program now running. It was refusing to accept the changes we made and failing to compile them. Our central belief is that the errors originate from where the menu bar was created, appBar.tsx. One method that we were unable to test was to encase each <MenuItem> in the file with a <grid Container>, so it will dynamically adjust to the screen side.

During March, the Coronavirus had hit, causing confusion and delays in working on the project, as we all had to work out how to do college courses remotely. Working from home was challenging from difficulty in finding a quiet space with family around, staying motivated & focused outside a college environment, and occasionally local internet not being available. The significant delay was digitally getting back in contact with the other project members and working remotely together.

Various computer obstacles prevented the project members from independently making much progress, and without physically having access to the same computer, they have not been able to resolve errors and troubleshoot in real-time. Sheltering in place negatively impacted our IQP progress because of the inefficiency of not being able to see each other's computers, and the inability to seek campus provided technical support or help from members of the programming club & the community board.

By the end of April, we were finally able to resolve the initial obstacles from working from home and got together as a team. We ascertained the compiling issue and we attempted to reinstall ReportAid to learn why the running program gave compile errors. When trying to run again, new problems occurred such as; the importing of 'Web3' could not be resolved.

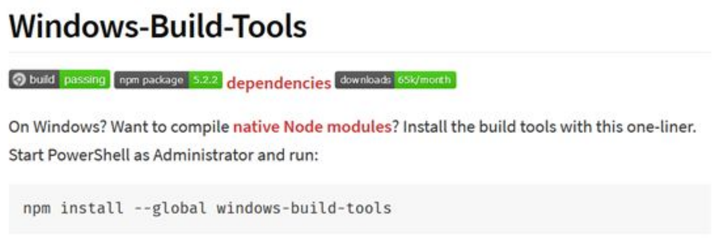


```
[./src/app/utils/history.tsx] 92 bytes {app} [built]
[./src/app/utils/strings.ts] 26.2 KiB {app} [built]
[./src/index.tsx] 1.01 KiB {app} [built]
    + 1517 hidden modules

ERROR in ./src/app/components/blockchain/blockchainProvider.tsx
Module not found: Error: Can't resolve 'web3' in '/Users/stephenlucas/ReportAid/
app/src/app/components/blockchain'
 @ ./src/app/components/blockchain/blockchainProvider.tsx 31:0-24 53:23-27
 @ ./src/app/components/blockchain/blockchain.tsx
 @ ./src/index.tsx
 @ multi @babel/polyfill ./src/index.tsx

ERROR in [at-loader] ./node_modules/@material-ui/styles/node_modules/jss/src/ind
ex.d.ts:8:78
    TS2344: Type 'Name' does not satisfy the constraint 'string'.
  Type 'string | number | symbol' is not assignable to type 'string'.
    Type 'number' is not assignable to type 'string'.
Child html-webpack-plugin for "index.html":
     1 asset
    [./node_modules/html-webpack-plugin/lib/loader.js!./src/index.html] 2.92 KiB
 {0} [built]
    [./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 489 b
ytes {0} [built]
```

Figures 5: Example of Web3 Missing

Recommendations for the Next Team

As was mentioned earlier in the report, it is necessary to learn and install the program's languages: JSX, TypeScript, and Material-IU. We recommend the next group resolve possible issues and delays with installing ReportAid on Linux or Mac. We think the program is more compatible with Windows because npm has made the Windows-Build-Tools package.


Windows-Build-Tools

On Windows? Want to compile native Node modules? Install the build tools with this one-liner. Start PowerShell as Administrator and run:

```
npm install --global windows-build-tools
```

Figures 6: Command to Install Windows-Build-Tools.

Another suggestion would be not to either do this IQP in only one term or finding a method to get a quicker response from the project proprietor/ primary programmer, Steve Huckle. We recommend doing it over at least three terms as we did. As discussed, there can be a week delay when messaging the manager because this is his side project. The next group could also consider improving the Report Aid's GitHub documentation to better explain the steps to install the program and how to use it. Huckle had also been looking into adding a logging/error reporting feature in the app so users can report their issues in the app instead of going to Github.

Conclusion

The main goal of the IQP was to contribute to an ongoing non-profit software project to gain experience providing real-life team projects that benefit society. The project we chose from Github, ReportAid, meets all criteria. The main focus was to fix UI issues, specifically fixing an issue related to the menu bar. The project requires knowledge of Git, JavaScript XML, TypeScript, & Material-UI, learning these languages was essential to fix the problem. We felt that our contributions to the project had been helpful but limited because of difficulties in installing the program. During the first term of the IQP, we thought we would install a working version of the program at the end of winter vacation. Because of the bugs we encountered across all three terms, we currently don't have a working version of the code, however, we helped with the processes of installation and we hope this will allow future teams to progress.

References

1. Mary K. Pratt, “How to Write a Statement of Work”, 22 MAY 2006, Accessed 9 November 2019

http://www.computerworld.com/s/article/111327/How_to_Write_a_Statement_of_Work

2.https://www.brighthubpm.com/project-planning/55260-writing-an-effective-statement-of-work/

3.Steve Huckle ReportAid Abstract,

https://github.com/glowkeeper/ReportAid/blob/master/docs/abstract.md. Accessed 9 November 2019

4. Postner, Lori & Burdge, Darci & Jackson, Stoney & Ellis, Heidi & Hislop, George & Goggins, Sean. (2015). Using humanitarian free and open-source software (HFOSS) to introduce computing for the social good. ACM SIGCAS Computers and Society.

45. 35-35. 10.1145/2809957.2809967.