Implementing Recommendations for the Labor and Delivery Application Utilizing Prototype Testing

by

Michael Krebs

April 25, 2019

Project Sponsor:
Amir Mehdizadeh, MD
UMass Memorial Hospital

Project Advisor:
Bengisu Tulu, PhD
Worcester Polytechnic Institute

IQP BXT-1803

Abstract

The goal of this project is to create a minimum viable product for the labor and delivery team at UMass Memorial Hospital based on the recommendations made by *Improving Labor and Delivery Tracking App*. A number of development frameworks and JavaScript libraries were compared to determine the best platform to develop the web application in the time frame of the project. We expect that the new prototype web application will serve as the foundation for a future project to build upon and eventually integrate into the labor and delivery team at UMass Memorial Hospital.

Acknowledgments

I would like to acknowledge the following people for their assistance and guidance that made this project possible. Dr. Amir Mehdizadeh, for his guidance and expertise in the labor and delivery field which enabled first-hand experience into how best this application can serve its potential users. Professor Bengisu Tulu for her support, passion and direction for the duration of the project. Lowe, Nawab, and Servidio, whose detailed work in evaluating and testing the prior application led to tangible recommendations that serve as the foundation for this project.

Table of Contents

List of Figures

List of Tables

Executive Summary

This project aims to improve upon the accuracy and efficiency in how labor and delivery patient information is recorded and accessed. To achieve this goal a minimum viable product was developed by integrating recommendations from prior projects. This project was organized under two major phases.

The first phase of the project was learning and building upon the previous application that was developed by (Curtis, Friedlander, Gelinas, Hammer, & Perullo, 2018) in time for the work of *Improving Labor and Delivery Tracking App* to conduct a field test that would provide insight to making a successful application. During this phase the focus was on developing the prior application and learning its intricacies as the original plan was to iteratively build upon the prior application until it was a viable product that the UMass Memorial labor and delivery ward could integrate. However, in accordance with the original developers' recommendation we switched the aim of the project to perform throwaway prototyping and create a new application based on what was learned from the previous application. In concurrence with the work of (Lowe et al., 2019) a proposal was created for how the new prototype application will be developed, integrating the recommendations made by both Dr. Amir Mehdizadeh, prior MQP and IQP teams and by Dr. Bengisu Tulu. I researched which frameworks and JavaScript libraries would be best to develop the application in the given the time and resources available for the project. During the formal proposal presentation to Dr. Amir Mehdizadeh and colleagues, we determined the goals for the next phase of the project.

The second phase of the project was to take what was learned from the previous phase and use this information to develop and test a new prototype. The prototype would be in a more modern JavaScript library that is expected to continue to grow in usage that will allow for easier maintenance and extensibility. The prototype best attempted to implement the three major recommendations from (Lowe et al., 2019) -design for experiences, design for context adaptability, and design for workflow integration - as well as incorporate the correct terminology and features that were explicitly stated as desirable by Dr. Amir Mehdizadeh. I used the React JavaScript library to develop the front end portion of the application with NodeJS and Express as my backend server and deployed the application to Heroku. After developing the application, I also developed a highly descriptive documentation, so that future work on the project can realize the end goal of integrating into the work by clinicians at the labor and delivery ward. I also provided recommendations for how the future teams working on this project should focus their efforts.

1. Introduction

In an era with rapid technological improvements, there are still areas of industry where the advancements have not been implemented. The healthcare industry is integrating technologies every day on large scales. However, some parts of the healthcare industry like labor and delivery wards which have specialized needs that are not always satisfied with enterprise systems are still looking for innovative technology solutions to help them improve their workflow and communication challenges. In 2017, the first iteration of this project was started with a partnership between a WPI MQP team and one of the residents working at UMass Memorial Hospital's labor and delivery ward. The work of (Curtis et al., 2018) created a mobile application that would help residents and attendants in the labor and delivery ward effectively record and monitor patient information. One year later the second iteration of this project started with the work of (Lowe et al., 2019) by getting a firsthand look into the work of the clinicians at the labor and delivery ward. With their firsthand insight and extensive research into the causes of error in medical fields they were then able to pinpoint why this application would be of value to the clinicians since miscommunication was a primary factor. They then evaluated and tested the application built by (Curtis et al., 2018) to see how it addressed the needs of all clinicians in the labor and delivery. Their work resulted in recommendations for future development which stands as the basis for this third iteration of the project.

Although miscommunications and absences of important information will always occur to a certain extent due to human error, this project aimed to develop a technology solution to facilitate seamless and timely communication among team members to minimize such instances in the labor and delivery wards. The primary objective of this project was to take the recommendations from the work of (Lowe et al., 2019) to develop a minimum viable product web application that serves as the foundation for future work to build upon and become an integral part in labor and delivery wards. The recommendations that this application addressed are (1) to design for experiences, (2) to design for context adaptability, and (3) to design for workflow integration.

2. Background

2.1 UMass Memorial Labor and Delivery

The sponsoring organization for this project is the labor and delivery ward at UMass Memorial Hospital. Currently, the transfer of information from clinician to clinician is done by reading patient information on a handwritten whiteboard, personal paper notes, and the electronic health record (EHR) system that they have in place. These mediums are often not up to date due to the fast pace environment in the labor and delivery wards, and provide inefficient forms of communication about the patients. The sign off process between shifts takes about 2 hours and includes a verbal debriefing of the patients where despite providers' best efforts miscommunications and absences of important details can emerge. The whiteboard and EHR system must be updated by the providers regularly. However, due to long shifts and fast paced environment, often times this updating occurs long after the events and information was new or relevant. Communication errors can have significant effects on patient care, therefore, accurate and effective communication in hospitals is a top priority.

2.2 Development Platform Considerations

Before beginning the development of the application serious attention was made to which development platform would be best given the constraints of the project. The main constraints of the project were the shortness of time to develop the application (6 weeks) and the need to continue building upon after the completion of this project. With these two constraints in mind, I needed a fast-developing framework that was well established and scalable.

The previous project (Curtis et al., 2018) utilized the Ionic framework to develop their application. This section will first discuss the reasons why this team chose the Ionic framework and the shortcomings it had when it came to further development and then discuss the new frameworks that were considered for development.

2.2.1 Ionic Framework

The original application, L&D Tracking App, was developed using an Ionic platform. The rationale behind using Ionic was that the application could be compiled to be both compatible with Android and IOS with only one version. A web browser preview of the app could also be used for fast prototyping and testing. The structure of the language is similar to HTML and the logical syntax was built on a modified version of JavaScript called Typescript; both of which the original team felt comfortable working with. Ionic was used primarily to develop the prototype rather than creating a full scaled deployable application (Curtis et al., 2018).

On the server side, the original project utilized a dedicated UNIX machine to host the application and database. The database was written in MySQL and stored basic information about patients, physicians and exams. They built an application program interface (API) by using Nodejs with a MySQL plugin in and Angular syntax (Curtis et al., 2018).

As mentioned by the original development team future implementations of the applications should “abandon the ionic framework.”(Curtis et al., 2018) Although the platform was well equipped for quick development and backend integration, it is not easily deployable to a full scale application as it fails in categories like user experience, notification handling and syntactical structuring of code. Moreover, during the first phase of the project, I had to pick up and learn a highly complicated undocumented program. Due to Ionic’s lack of clear syntax and integration issues, workarounds and technological debt accrued to a point where further developing the application required constant reach out to the original developers who have since moved on from the project.

For the server portion of the project, utilizing the dedicated UNIX machine also proved to be a challenge for developing. Although a local version of the project could be used for rapid development, having to push the project to the virtual machine often caused unforeseen breaks and errors that were difficult to debug since the virtual machine only had terminal capabilities. This was particularly troublesome when working with proxied addresses for server querying during production stages. It is evident the prior project also experienced trouble with this aspect of the project based on comments and error logging.

2.2.2 Angular

Angular is an older JavaScript library, and its usage and popularity has been declining (TechMagic, 2018). Angular was in part used during the previous application and unfortunately did add to some of the complications as there were two platforms being used concurrently. Although its documentation is comprehensive and well tested, Angular was not easy to pick up immediately while I was attending to the old application in phase 1. The documentation for Angular is comprehensive but unintentional naming conventions means picking up and developing in Angular would be time consuming (Google, 2019). Given that Angular’s use is in decline and the goal of the project is to be built upon, continuing to use Angular was ruled out.

2.2.3 Vue

Vue is a newer JavaScript library with a lot of promise and growing in popularity. Vue is an advanced JavaScript library that provides a lot of flexibility when it comes to development (TechMagic, 2018). Upon researching there was not a lot of examples on how to integrate Vue with an API or backend server, particularly with MySQL. There are many programmers who find that the learning curve for Vue was not as steep as React and thought that the design was simpler to understand as it felt like HTML. But it is also worth noting that since it is a newer library there are not as many compatible third-party libraries that speed up development time. Vue is also not widely used in the professional world (Tarnowski, 2017). With these limitations Vue was not chosen to be the framework to use.

2.2.4 React

React is a fast-developing JavaScript library that prides itself on being intuitive and easily implemented. Adding React to a JavaScript program is as simple as importing the library and adding one line of code above and below your existing project (Facebook, 2019b). React is also still growing in its usage which lends itself nicely to development down the road when a future group builds upon the application (Buna, 2017). React is easy to learn in large part because of its simple design, reusable components, and use of JSX which is similar to HTML that most programmers have experience with and can be read easily. React also has detailed documentation that lends itself well to solving issues quickly and finding examples on how to do what you want to do (TechMagic, 2018). Another perk to React is that it comes with great add on packages like create-react-app which creates a boilerplate program that has deployable and local development ability right from the start (Facebook, 2019a). This project is solely focused on creating a web application, but should future work decide to go platform specific, React Native is quite similar to React so going platform specific would not require much work. In fact the skills and basic structural learning curve you gain from developing in React translate well to React Native (TechMagic, 2018). The reason this is enticing is because React Native can be compiled into both IOS and Android which the last project indicated as the two major native platforms that would need to be covered in order for this application to be better used on mobile devices (Facebook, 2019c). Since React was the best option for both the immediate work on this project and for future work I decided to choose React at the JavaScript library to create this app in.

2.2.5 Express and NodeJS for Server Side

The only remaining portion of the project was to determine how the backend part of the project would work. Since there already existed a well-defined database, it did not make sense to move away from MySQL. However, an API still needed to be developed and I had little experience in doing so. Upon researching, Express and NodeJS were coupled together for creating the server that would host the database. I found many examples of how to create a server using create-react-app express and Nodejs but followed the work of (Moa, 2018) when developing the application. The previous application also used NodeJS, so I already had exposure in phase one of the project which was invaluable to hit the ground running in phase two. Express and NodeJS was implemented as the server portion of the project and would be queried from within the application at a proxied address to handle all the database integration. Other middleware was also used like Morgan and Cors for security reasons as to not expose the database to the user.

2.3 Deployment Using Heroku

Given the time frame of the project we decided to use an existing deployment platform to speed up the development. Through research it was found that Heroku was easily integrated with React. Heroku has a build pack that specifically integrates with create-react-app, a boilerplate addon for building React applications. Using this deployment service, the application could easily be deployed and managed by outsourcing the complexities for hosting. I utilized the step by step guide from (Hall, 2019) on how to run the build pack on a create-react-app project. During the scenario testing of the application by (Lowe et al., 2019) they did experience technical difficulties that could have been attributed to stress on the server. With Heroku, the application is now in a deployable environment that when performing a similar test would cause no stress to the server, making it more reliable. Since this project is designed to lay the foundation of a web application and be a prototype, outsourcing the hosting of the application proved to be a time saving and invaluable step in developing the application quickly.

3. Methodology

3.1 Project Initiation and Scoping

To develop a usable and effective app, it was essential to understand the preferences and requirements for all the clinicians in the labor and delivery ward at UMass Memorial. Prior to my efforts as a developer, a project team researched and prioritized the needs of those clinicians so that a sufficient app could be scoped. It was made clear through their research that above all else, clinicians wanted an app that would be user friendly so that it didn't interrupt their work flow.

The app prototype before mine was used to test usability, effectiveness of EHR technology in the workplace, and understand major roadblocks with app integration. It was determined that this early prototype app was promising but not sufficiently developed and further development using the Ionic platform was not feasible, however it was incredibly important in gathering information necessary to scope the app that I developed for my project. Through wireframe testing, surveys, and "real life" scenario simulations, the group was able to obtain valuable information about design and feature specifics as well as quantitative data about the usability of the app. The Systems Usability Survey yielded results reflecting that the ease of use was quite poor; 54% (+7%) of respondents agreed that the app was not very usable (Lowe et al., 2019). Because of the rigid nature of Ionic framework, users were unable to fluidly access their data which poses a large roadblock for clinicians in the Labor and Delivery Ward. To improve this essential component of an effective app, it was determined that another platform must be used.

In addition to quantitative data, the group also collected significant details about the design and necessary features. This includes medical terminology and abbreviations, drop-down vs free text preferences, important prompts, as well as major concerns and potential roadblocks. All this input helps to construct an app that addresses as many requirements as possible. From the previous groups research, my design for the app hopes to improve flow and usability while eliminating the major roadblocks and "glitches" that came with the previous prototype.

3.2 Requirements Gathering and Prioritization

The requirements for the new application were derived mostly by the work of (Lowe et al., 2019) who, after asking residents and clinicians at the labor and delivery ward during their scenario testing, created tables to highlight some of the new features that the clinicians would like to see in a future application. The second portion of the requirements came from Dr. Amir Mehdizadeh. He and I exchanged emails and documents that would highlight and clarify the terminology and features he and the rest of the labor and delivery ward wanted to see in the application. Tables 3, 4, 5, and 6 found in the Appendix contain, the recommendations, desired fields and elements to be included in the application.

One of the most important components in this application being successful is that it makes the jobs of the clinicians easier, not harder to use. Regardless of how it could improve communication if no one uses the app then it serves no purpose. With that I had to prioritize which features I would implement in the new application given my time frame to work on this project as I would not be able to implement them all. The most important aspect I considered while determining which features to implement was functionality and usability. The application had to do what the clinicians wanted the app to do and it had to be intuitive in how they can get that done. The features requested about stylistic, aesthetic and design were put on the backburner of the project as things to implement should all before them be completed. The prioritized features can be found in Table 1.

Table 1 Prioritized Features

Requirement	Priority	Complexity
All Patient Page Functionality	1	Medium
Subscribe/Unsubscribe ability	1.5	Medium
Add Patient Page Functionality	2	Medium
Add Event Page Functionality	3	Easy
Login Page Functionality	4	Hard
Medical History Page Functionality	5	Medium
Whiteboard Page Functionality	6	Medium
All Patient Page One Liner	7	Medium
Navigation Between Pages	8	Medium
Edit Patient	9	Medium
Edit Event	10	Medium
Edit Medical History	11	Medium
Style Application	12	Hard/ Time consuming

One issue that arises when working across industries is that often times the language and terms are different and the understanding of the other industry is generally minimal. As a software developer it is one’s goal to implement all the features that are asked but sometimes such features are not possible or would be better implemented in other ways. To close this gap between industries once the final list of prioritized features, that both came the requirements shown in Table 1 and my realistic evaluations/modifications of them, were consolidated at the end of phase one of this project I presented my proposed features to implement to Dr. Amir Mehdizadeh and colleagues at UMass Memorial Hospital for approval. At the end of the presentation we held a discussion portion where aspects of the features were fine-tuned and I was ready to begin phase two of the project.

3.3 Development Methodology

The developed methodology that I utilized was iterative development. Each week I created task lists for what features I was going to implement and would then present my progress for that week to my advisor Dr. Bengisu Tulu. In these meetings we would discuss the current state of the app and discuss what features we should tackle in the upcoming week, at times deviating from the order of the priority list. This approach enabled the development of the app to be adaptable because instead of seeing the requirements as a contractual obligation I used them as the basis on which to develop the app. In doing so I was able to, when developing the application, see that some of the requirements would be better in different views of the application or implemented in different ways.

4. Results

4.1 Documentation of App

4.1.1 Overview

This application has three major components: the front end React components (client), the API and the server. Figure 1 shows the flow of information between the user and the application.

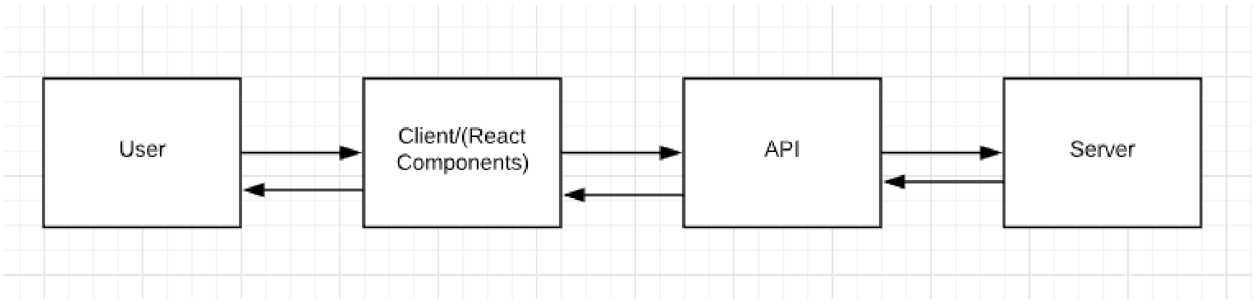


Figure 1 User Application Information Flow

With this flow of information, the user cannot access the database or interact with the API directly. This is a security measure intended so that the user can only access the information that is given to them by the application. Each of the three major components will be addressed in detail in the next subsections.

4.1.2 Client

4.1.2.1 Client Overview

The client portion of the application is where most of the project lives. The client is responsible for taking in interactions from the user, displaying the correct information and retrieving the information that is needed via asking the API. In many React applications it is a simple one-page application. Although this project is still only one page, it mimics that of a multi-page application via React Routing (more on React Routing in subsection 4.1.2.2). The client is composed of one main component and many other subcomponents.

The main component, called App.js, is the root component of this project. It is the first component that gets loaded and displayed. The main responsibility of this component is to import and announce all the other react components that this project will be routing to. For this project the App.js can be found in the “client/src” folder. This main component is established as the root component by another JavaScript file, index.js in the same location. Figure 2 shows the line of code where App.js is set as the root component within index.js.

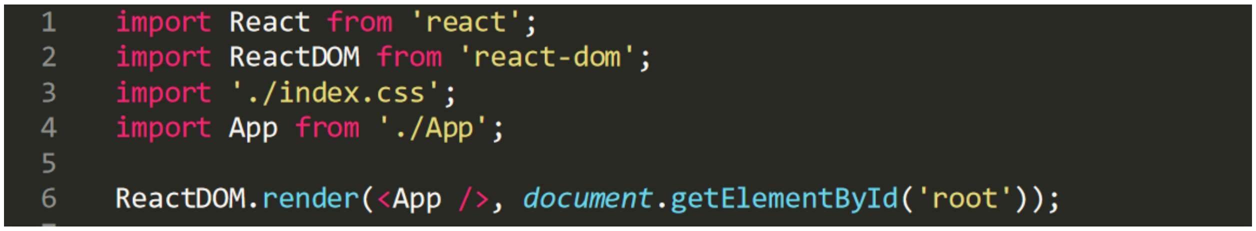
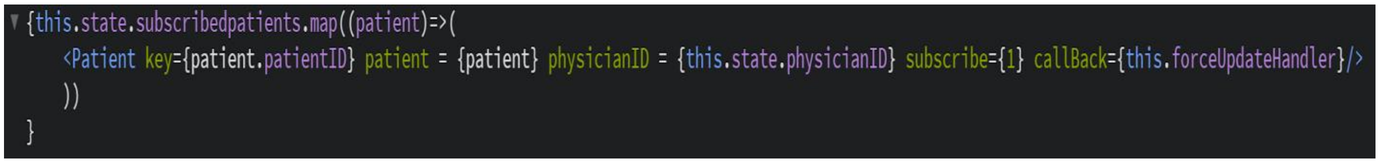

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

Figure 2 Root Component

All of the components import React, but only this file imports ReactDOM(other components can but none of the functions within ReactDOM were needed for this project). The main portion of this snippet of code is to show how index.js imports App from '.App' where App is referencing App.js. And then when the render method is called saying "Render the App component and call it root."

The other subcomponents are not like subclasses within an object-oriented hierarchy. Instead they are just as "powerful" or can be much larger than the main component. The main component is simply special because it is loaded first and has the responsibility of importing and announcing the components the project will use. But these subcomponents, for lack of a better term, also must import other subcomponents when they want to use them. All these subcomponents are found in the "client/src/components" folder. Each of the components has their own respective folder in the components folder and these folders contain the component JavaScript file, sometimes a ".CSS" file or other components that are used to help create a component. Components should not be thought of as html pages but rather like a <div> tag. Components are used in composition with other components to create a page. A page in a React app does have a different URL Figure 2 demonstrates how a page is composed of many components. Each rectangle in the Figure 2 is a component, including the blue rectangle that contains the 9 inner red rectangles. It is vital to understand that breaking a page up into components is cleaner and creates a better designed project.

A component has one major function that always needs to return, or you will get an error. This function is the render function. A component needs to render, or it will be of no value (you can return null but that again makes the component useless). Within this render function you are coding in what is called JSX. JSX is just like HTML except you can inject JavaScript directly into it by placing the JavaScript inside of curly brackets {}, as seen in Figure 3. The inner red rectangles within the blue rectangle are created by mapping through an array of patients and for each patient in that array we create an inner component of patient information. For now, ignore what comes after "<Patient" just see how within the curly braces I injected JavaScript to call another component multiple times.



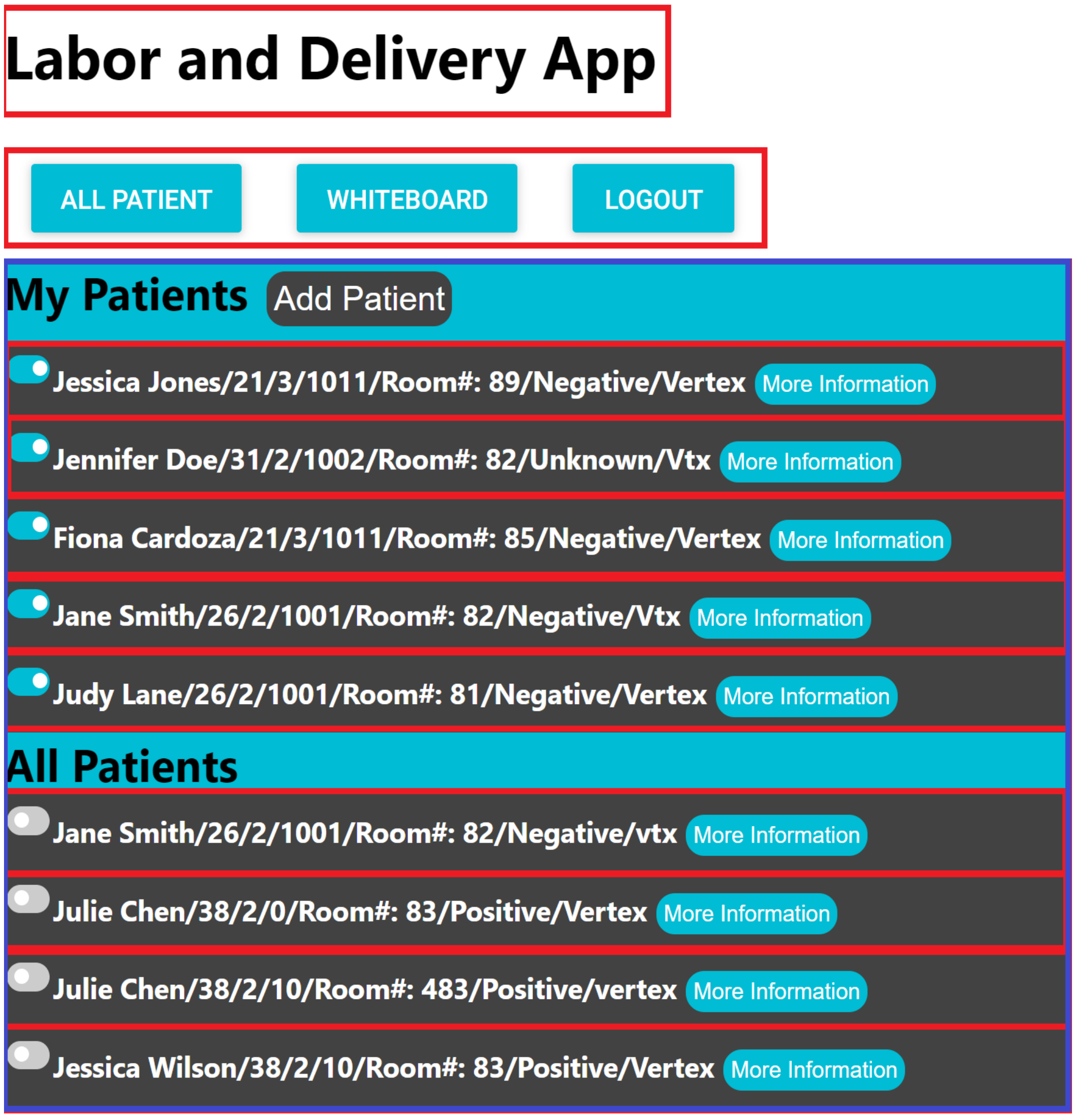
```
{this.state.subscribedpatients.map((patient)=>(
    <Patient key={patient.patientID} patient = {patient} physicianID = {this.state.physicianID} subscribe={1} callBack={this.forceUpdateHandler}/>
    ))
}
```

Figure 3 JavaScript Injection in JSX

The last important concept to understand about components is that they keep variables in its "state." Every component has a state, although you do not need to code a state if you are not going to use it. The state can simply be seen as instance variables that are accessed within the rest of the component using "this.state.*variable_name*." You can pass along these state variables from parent to child components and are invaluable when composing components together.

4.1.2.2 Client Architecture

The client portion has a sequence in which the components are navigated to and from. The most readily used component is the AllPatient component. This was the marked with a dark blue rectangle in Figure 4. There are two other components that are part of every page: the Header component and the NavBar component. These two components marled with the top two red rectangles in Figure 4. Figure 5 shoes the interface structure diagram which displays how a user can navigate the application. Note that each rectangle in Figure 5 represents a page, and the lines connectiong each rectangle rpresents the routing between pages. However, the lading page for our application is the Login page. This page is comprised mostly of the login component. This component handles the username and password authentication. Currently the application simply takes the username and password, determines if a user in the database has that username and if so it checks to see if the passwords match. Once authenticated the corresponding physician ID accompanied with that user is passed to the AllPatient page.


Labor and Delivery App

My Patients

Jessica Jones/21/3/1011/Room#: 89/Negative/Vertex

Jennifer Doe/31/2/1002/Room#: 82/Unknown/Vtx

Fiona Cardoza/21/3/1011/Room#: 85/Negative/Vertex

Jane Smith/26/2/1001/Room#: 82/Negative/Vtx

Judy Lane/26/2/1001/Room#: 81/Negative/Vertex

All Patients

Jane Smith/26/2/1001/Room#: 82/Negative/vtx

Julie Chen/38/2/0/Room#: 83/Positive/Vertex

Julie Chen/38/2/10/Room#: 483/Positive/vertex

Jessica Wilson/38/2/10/Room#: 83/Positive/Vertex

Figure 4 Component Composition

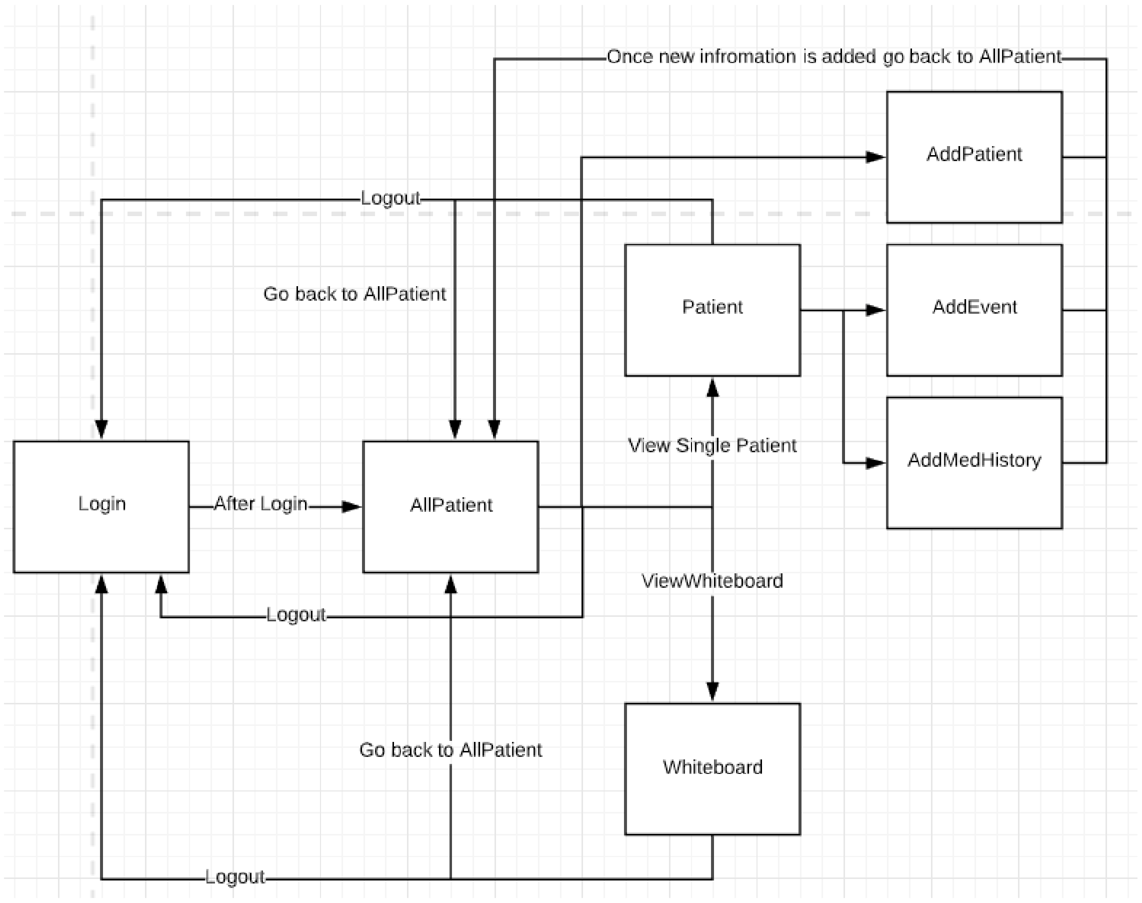
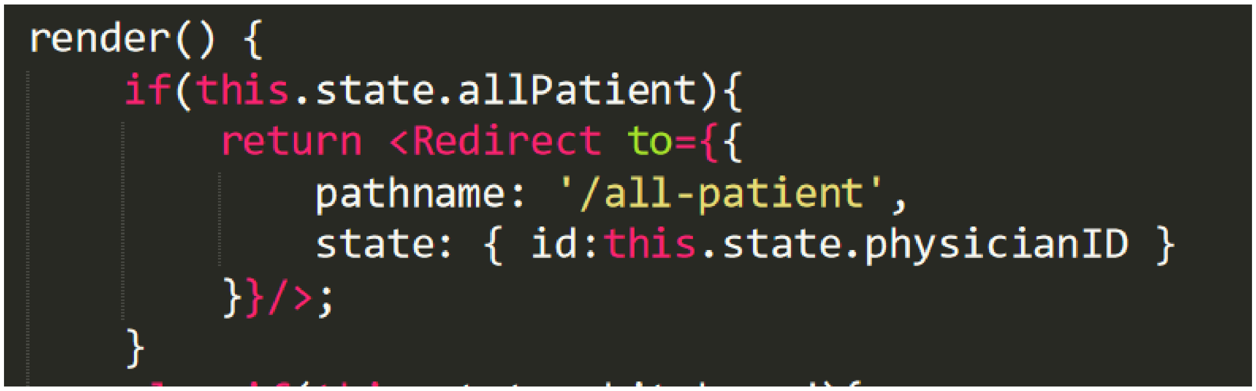


Figure 5 Client Interface Structure Diagram

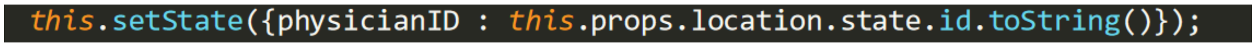
The AllPatient page displays all the patients that the user is subscribed to at the top of the pafe and all the patients they are not subscribed to at the bottom section of the page. This page also facilitates subscribing and unsubscribing to patients. The navigation bar is the main way of going between different pages but to add new information to the database (AddPatient, AddEvent, AddMedicalHistory) you need to click the corresponding button on the AllPatient page. That is why this page is seen as the main page of the application.

The way to route from one page to the next is by returning another page in the render method within a component. The easiest I came across was to use Redirect, which is an extension to React. Redirect simply redirects the user to a different component. Within Redirect you can specify the variables you would like the new page to have once it loads. This is done by passing a state, see Figure 6. Within the new component where you want to make use of these variables you call the variable by saying "this.state.props.location.*variable_name*." Example code for both sending variables and receiving them can be seen in Figures 6 and 7.



```
render() {
    if(this.state.allPatient){
        return <Redirect to={{
            pathname: '/all-patient',
            state: { id:this.state.physicianID }
        }}/>;
    }
```

Figure 6 Sending Variable with Redirect



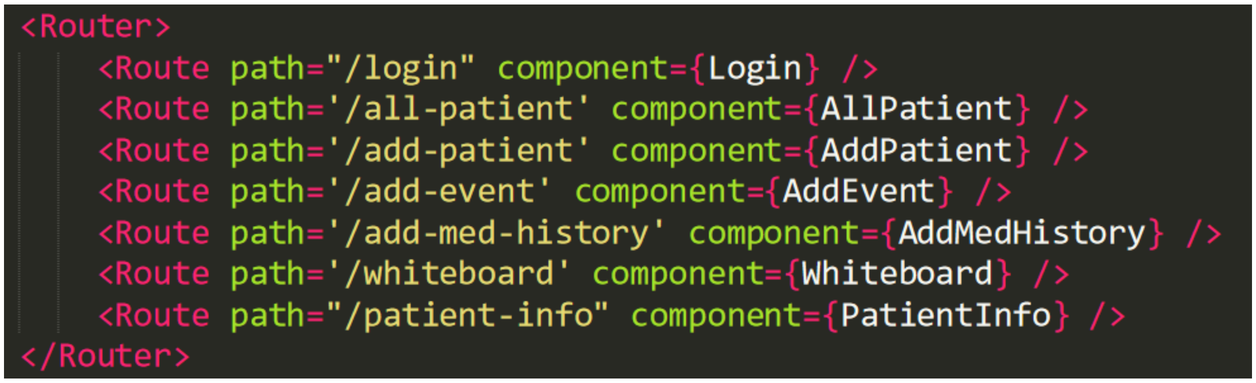
```
this.setState({physicianID : this.props.location.state.id.toString()});
```

Figure 7 Receiving Variable within Desired Component

From Figures 6 and 7 you can see that we passed along a state with a variable called "id" that was set to "this.state.physicianID" and within the new component we set "this.state.physicianID" to "this.props.location.stat.id.", which syntax aside just means we copied the old "physicianID" from the previous component's state to the new component's state we are now working in. In this project, we need to constantly pass around the "physicianID", therefore, this was done frequently throughout the program.

4.1.2.3 Creating a New Page

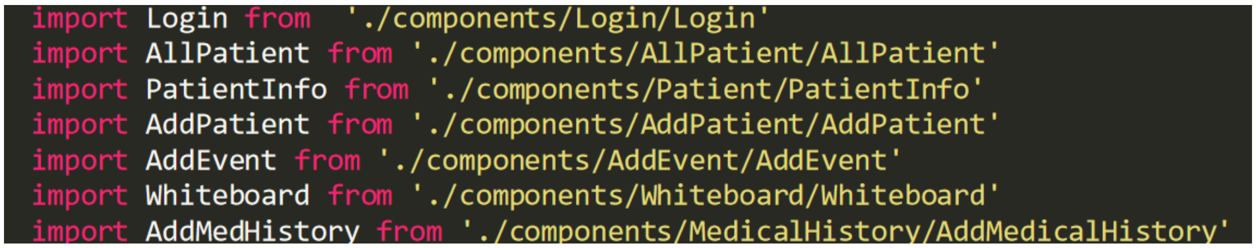
To create a new page all you need to do is add the Route in App.js and create a component that will be called when that new route is navigated to. Figure 8 shows the current routes in the project and subsequently where a new page would need to be added.



```
<Router>
    <Route path="/login" component={Login} />
    <Route path='/all-patient' component={AllPatient} />
    <Route path='/add-patient' component={AddPatient} />
    <Route path='/add-event' component={AddEvent} />
    <Route path='/add-med-history' component={AddMedHistory} />
    <Route path='/whiteboard' component={Whiteboard} />
    <Route path="/patient-info" component={PatientInfo} />
</Router>
```

Figure 8 Routes within App.js

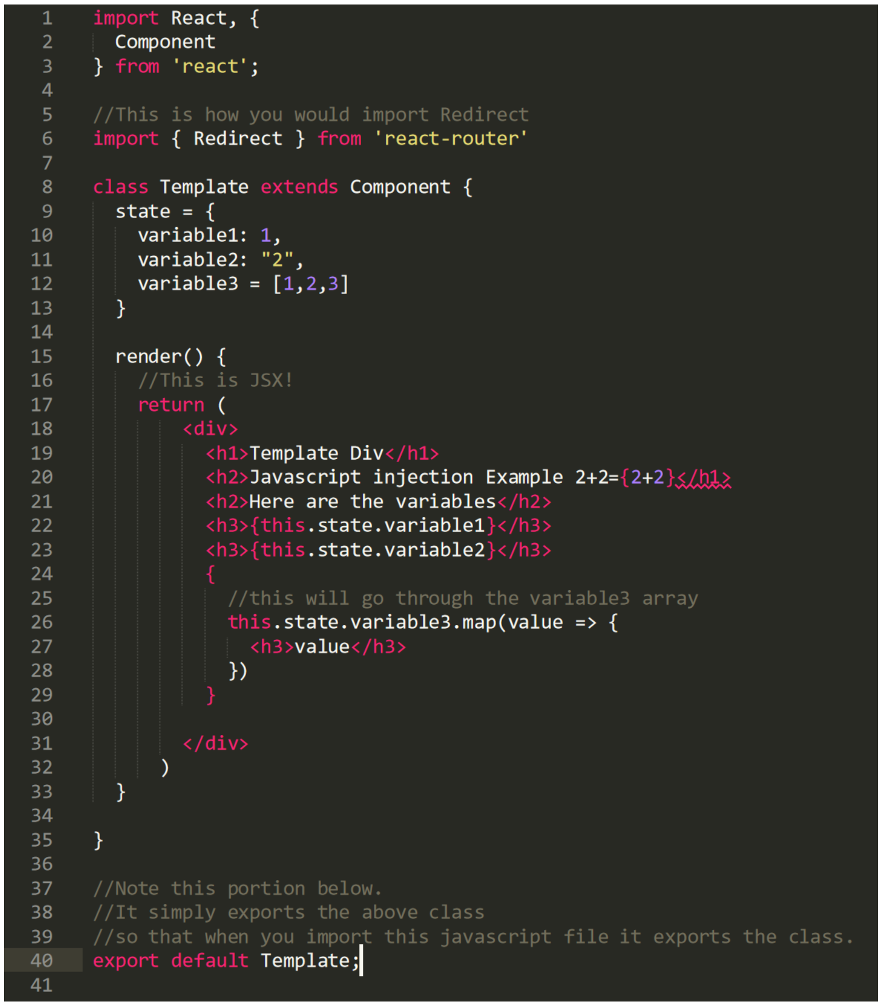
The path can be thought of as the URL and the component is the main component that you will want to be rendered. Notice how the component is within {} implying that this is JavaScript and that those words in white are actually variables of some sort. At the top of App.js you will find imports where we are importing the components themselves as can be seen in Figure 9



```
import Login from  './components/Login/Login'
import AllPatient from './components/AllPatient/AllPatient'
import PatientInfo from './components/Patient/PatientInfo'
import AddPatient from './components/AddPatient/AddPatient'
import AddEvent from './components/AddEvent/AddEvent'
import Whiteboard from './components/Whiteboard/Whiteboard'
import AddMedHistory from './components/MedicalHistory/AddMedicalHistory'
```

Figure 9 App.js Importing Components

From here on, all you need to do is (1) create a component, (2) make sure that the render function is returning some JSX and (3) make a route to it by including something like what you see in Figure 6 from the page that you want to redirect to your newly created page. A template component is show in Figure 10. This template component can also be found in by navigating to the template folder which is in the components folder.



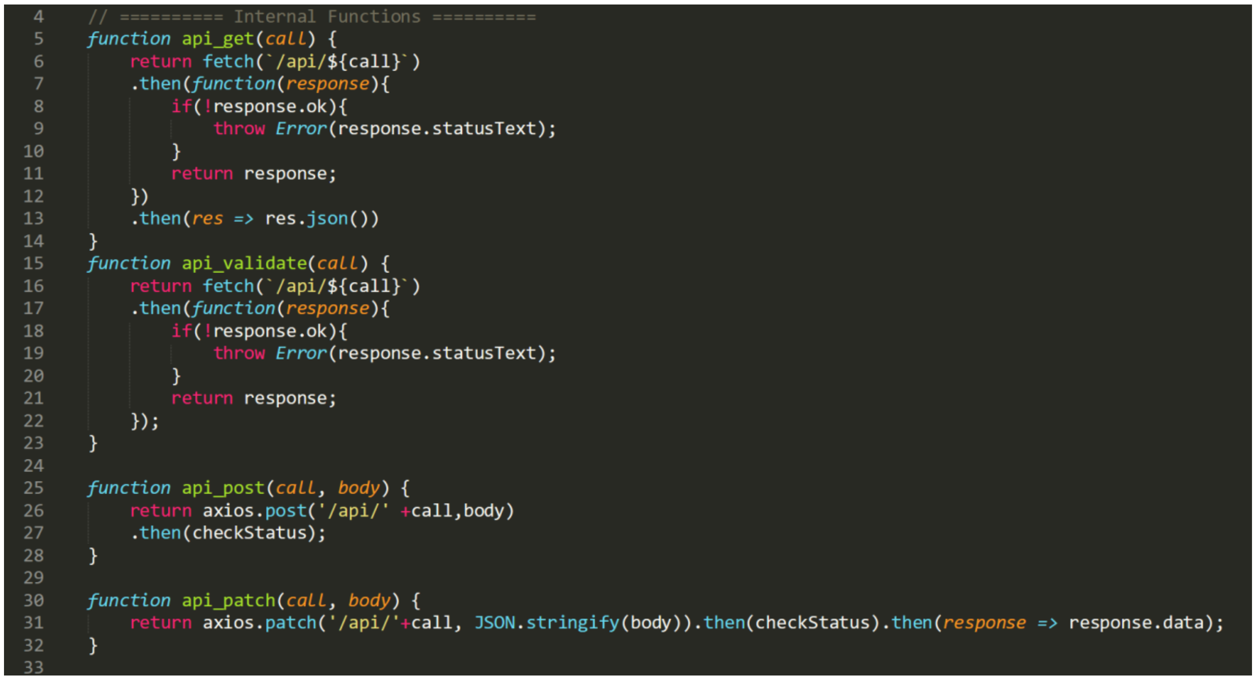
```
import React, {
  Component
} from 'react';

//This is how you would import Redirect
import { Redirect } from 'react-router'

class Template extends Component {
  state = {
    variable1: 1,
    variable2: "2",
    variable3 = [1,2,3]
  }

  render() {
    //This is JSX!
    return (
        <div>
          <h1>Template Div</h1>
          <h2>Javascript injection Example 2+2={2+2}</h1>
          <h2>Here are the variables</h2>
          <h3>{this.state.variable1}</h3>
          <h3>{this.state.variable2}</h3>
          {
            //this will go through the variable3 array
            this.state.variable3.map(value => {
              <h3>value</h3>
            })
          }

        </div>
      )
  }

}

//Note this portion below.
//It simply exports the above class
//so that when you import this javascript file it exports the class.
export default Template;

```

Figure 10 Template for Making a New Component

4.1.3 API

The API that was created is barebones and straightforward. There is only one JavaScript file that makes up the API and that is client>src>Database>DatabaseInteract.js. Within this class there are three types of functions that are broken up by comments, so they are clearly defined.

The first of these functions are the internal functions. These are the functions that make the requests to the server. There are three types of request, fetch, post and patch. So far, I have only needed to use fetch and post but in the future patch will likely need to be used. Note, that in this request to the server "/api" is prepended. We could have prepended it with any word so long as it matched what our server was expecting. It simply creates proxy addresses to our server so that we do not run into same port or unexpected issues when trying to host two separate programs (the client and server). Figure 11 shows the internal functions that already exist in the project.

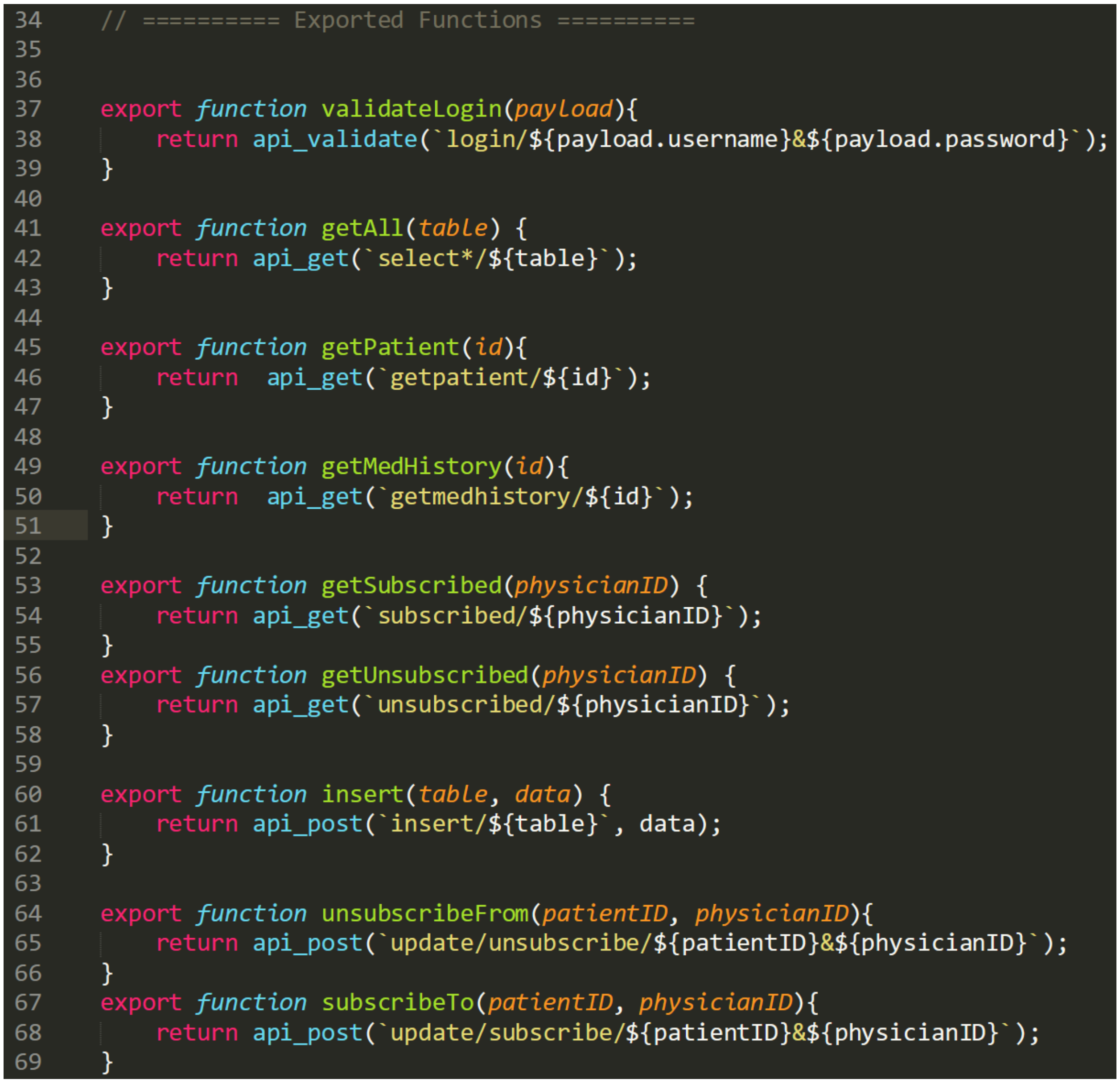


```
// ========== Internal Functions ==========
function api_get(call) {
    return fetch(`/api/${call}`)
    .then(function(response){
        if(!response.ok){
            throw Error(response.statusText);
        }
        return response;
    })
    .then(res => res.json())
}
function api_validate(call) {
    return fetch(`/api/${call}`)
    .then(function(response){
        if(!response.ok){
            throw Error(response.statusText);
        }
        return response;
    });
}

function api_post(call, body) {
    return axios.post('/api/' +call,body)
    .then(checkStatus);
}

function api_patch(call, body) {
    return axios.patch('/api/'+call, JSON.stringify(body)).then(checkStatus).then(response => response.data);
}

```

Figure 11 Internal Functions

In the internal functions, fetch is used for getting data from the server, post is for sending data and patch is for doing updates.

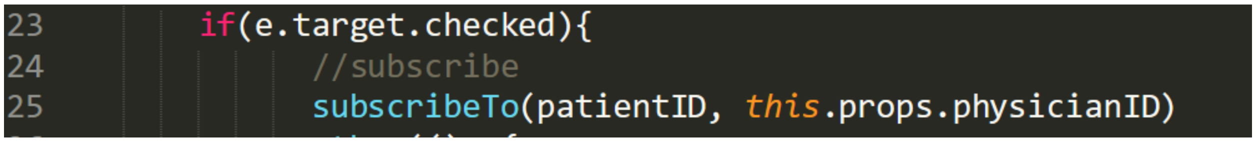
The second kind of functions is the exported functions. These are the functions that the components will call to get the information they need. These functions call the appropriate internal functions with the correct end point while pass along the needed parameters. Figure 12 showcases what the exported functions look like.



```
// ========== Exported Functions ==========


export function validateLogin(payload){
    return api_validate(`login/${payload.username}&${payload.password}`);
}

export function getAll(table) {
    return api_get(`select*/${table}`);
}

export function getPatient(id){
    return  api_get(`getpatient/${id}`);
}

export function getMedHistory(id){
    return  api_get(`getmedhistory/${id}`);
}

export function getSubscribed(physicianID) {
    return api_get(`subscribed/${physicianID}`);
}
export function getUnsubscribed(physicianID) {
    return api_get(`unsubscribed/${physicianID}`);
}

export function insert(table, data) {
    return api_post(`insert/${table}`, data);
}

export function unsubscribeFrom(patientID, physicianID){
    return api_post(`update/unsubscribe/${patientID}&${physicianID}`);
}
export function subscribeTo(patientID, physicianID){
    return api_post(`update/subscribe/${patientID}&${physicianID}`);
}
```

Figure 12 Exported Functions

Two examples of the exported functions that I will go into detail with are "subscribeTo" and "getPatient." The exported function "subscribeTo" takes in two parameters, one being the "patientID" and the other being the "physicianID". This function is being called in "AllPatient-Patient.js" as shown in Figure 13.



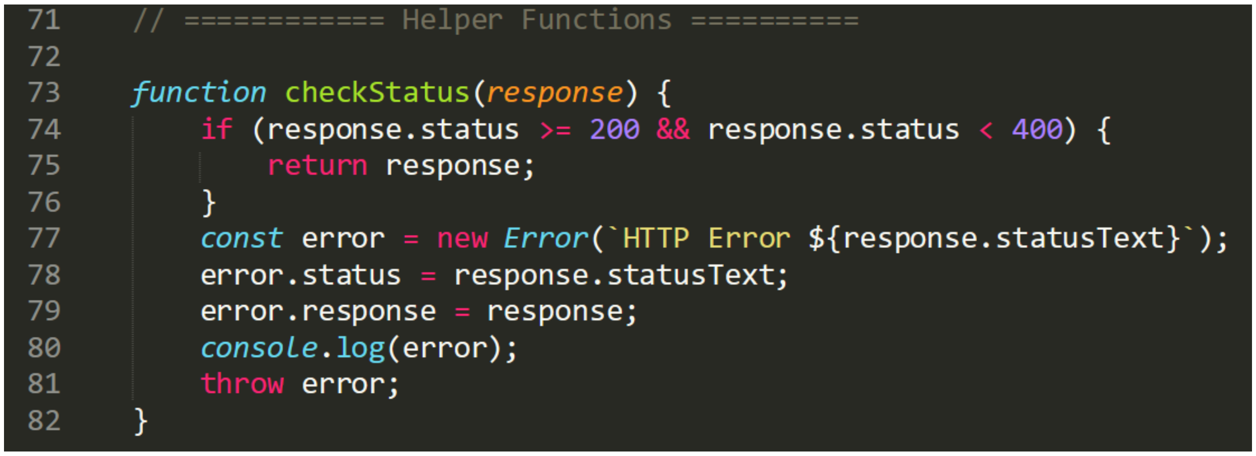
```
        if(e.target.checked){
            //subscribe
            subscribeTo(patientID, this.props.physicianID)
```

Figure 13 Exported Function Being Called

The variable “e” is the event handler for the slider checkbox. This bit of code then reads, if the slider was just checked then we call subscribeTo with the “patientID” and the “physicianID.”

Back to the API exported function we send these two values to the endpoint, notice the “&” delimiting the two variables, to the endpoint set up in the server (which is explained in detail in the next section).

The last kind of functions are the helper functions. These are not vital but, so they help maintain clean code. The only helper function currently is the “checkStatus” function which simply returns the response if the request was good or returns an error if it was bad. Figure 14 showcases checkStatus.

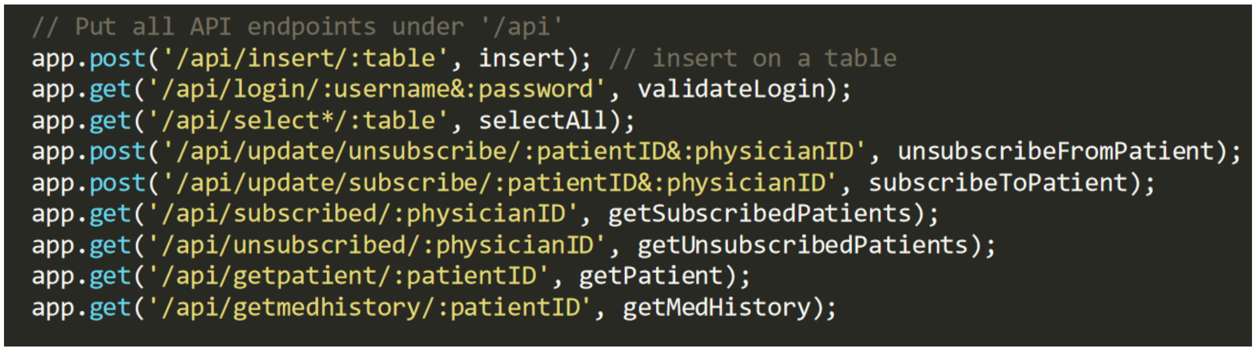


```
// ============ Helper Functions ==========

function checkStatus(response) {
    if (response.status >= 200 && response.status < 400) {
        return response;
    }
    const error = new Error(`HTTP Error ${response.statusText}`);
    error.status = response.statusText;
    error.response = response;
    console.log(error);
    throw error;
}
```

Figure 14 Helper Functions

4.1.4 Server

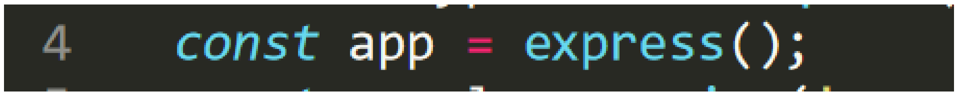
The server is a bit more complicated than the API but still very manageable. The server file is located outside of the client folder and within the project folder itself. The name of the file is server.js. The first thing to know about the server is how to create new endpoints. Figure 15 shows how end points are created.



```
// Put all API endpoints under '/api'
app.post('/api/insert/:table', insert); // insert on a table
app.get('/api/login/:username&:password', validateLogin);
app.get('/api/select*/:table', selectAll);
app.post('/api/update/unsubscribe/:patientID&:physicianID', unsubscribeFromPatient);
app.post('/api/update/subscribe/:patientID&:physicianID', subscribeToPatient);
app.get('/api/subscribed/:physicianID', getSubscribedPatients);
app.get('/api/unsubscribed/:physicianID', getUnsubscribedPatients);
app.get('/api/getpatient/:patientID', getPatient);
app.get('/api/getmedhistory/:patientID', getMedHistory);
```

Figure 15 Server Endpoint Creation

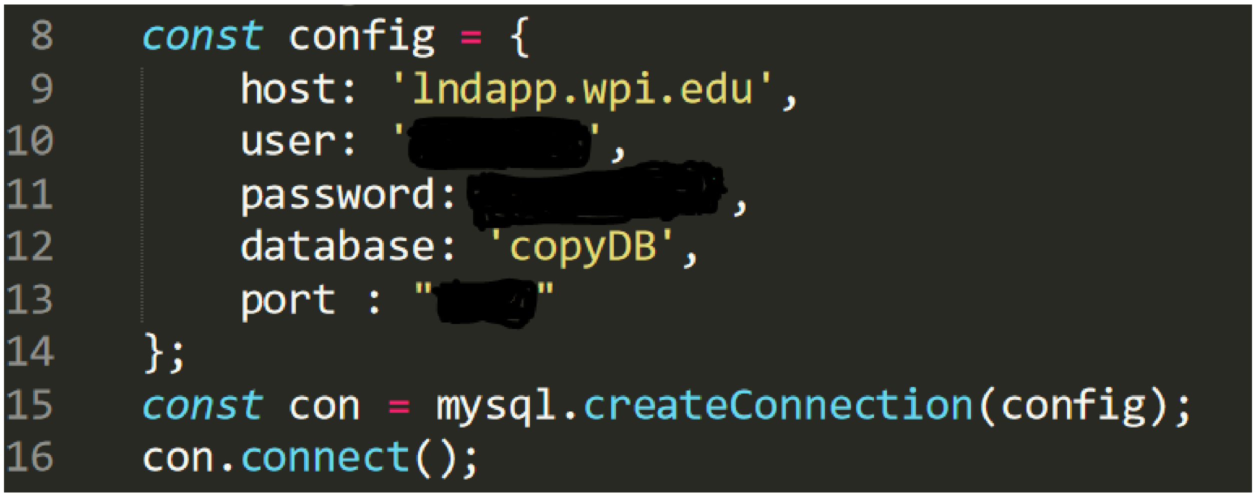
Notice how everything starts with “app”. The reason for this is because “app” comes from our back-end provider express from the line of code shown in Figure 16. It isn’t necessary to know what this is doing exactly but just so you are aware where the declaration is coming from.



```
const app = express();
```

Figure 16 App Declaration in Server

The next thing to notice is that we are setting all the endpoints by prepending “/api” this is consistent with what we did in the API. After that the middle portion is simply the endpoint and can be anything that makes sense. The more intentional the better. The last thing to know about end points is the “:” meaning. The colons represent variability with the endpoint and are going to be passed in as essentially parameters. For example, in the endpoint “/api/select*/:table.” Table in this case is hoping to get a table in the database and like the endpoint suggests we are going to select all from that table. After the comma is simply the method that you would like to run when this endpoint is hit and these functions are within the server.js file. But before we look at how that works we need to first see how we connected to the database in the first place. The first thing that needed to be done was import the “mysql” add on. From there we create the connection with our database by entering the name, username, password and port and finally connecting to the database. Figure 17 showcases all the steps to connect to the database beside importing “mysql” which is required.

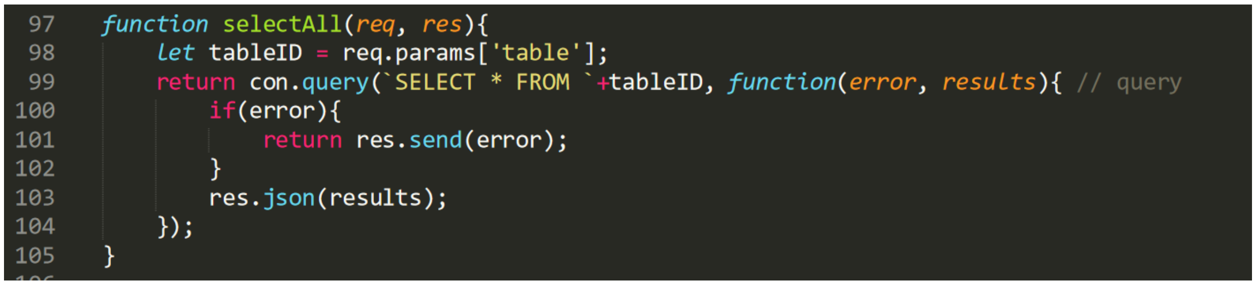


```
const config = {
    host: 'lndapp.wpi.edu',
    user: '[redacted]',
    password: [redacted],
    database: 'copyDB',
    port : "[redacted]"
};
const con = mysql.createConnection(config);
con.connect();
```

Figure 17 Connection to the Database

The const “con” is now connected to the database and we will be able to run queries on it easily.

Figure 18 shows how the “parameters” discussed with endpoints are accessed from with a function and how this function can query the database.

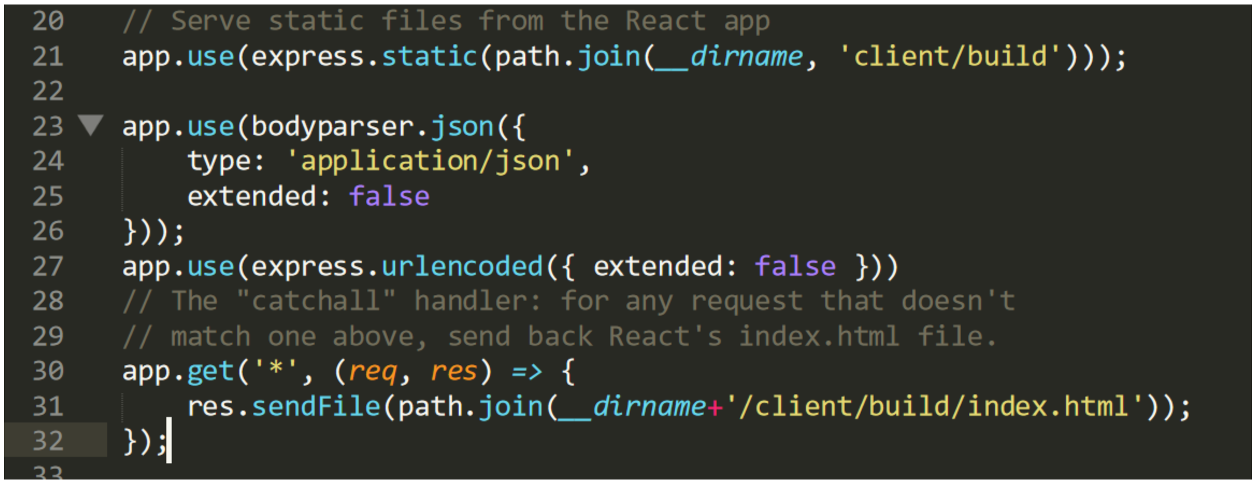


```
function selectAll(req, res){
    let tableID = req.params['table'];
    return con.query(`SELECT * FROM `+tableID, function(error, results){ // query
        if(error){
            return res.send(error);
        }
        res.json(results);
    });
}
```

Figure 18 Example Function Call in Server.js

The “req.params” statement is what is storing those parameters we got from the endpoint. And once we access it by specifying which parameter we want (“/api/select*/:**table**.” and “req.params[‘**table**’]”), we can use that like any other variable. The “con.query” statement is a function that queries the database where the first parameter is the query and the second is a place for you to process the results. Note how we first check to see if this request resulted in an error. If it did not, we turn the results into json and we return it. This return will go back to the exported function in “DatabaseInteract” which will return it directly to our component to use.

The lines of code in Figure 19 are for assisting the server to find the correct files when the app gets deployed. You do not have to worry about these.



```
// Serve static files from the React app
app.use(express.static(path.join(__dirname, 'client/build')));

app.use(bodyparser.json({
    type: 'application/json',
    extended: false
}));
app.use(express.urlencoded({ extended: false }))
// The "catchall" handler: for any request that doesn't
// match one above, send back React's index.html file.
app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname+'/client/build/index.html'));
});
```

Figure 19 Deployment Proxying Instructions

4.1.5 Development Server

To start the application locally and develop it in real time you need to do the following. First navigate to the project folder and run “npm start.” If you do not have npm installed, you need to do so. You should then see that the server is running. Once you see this go to the client folder and run “npm start” again. This time it will take longer but after it compiles will open a development window on localhost:3000. And you’re done! Every time you save the development localhost will refresh and you will still be able to interact with the database.

4.1.5 Deployment

Once you feel you have something worth deploying, navigate to the client folder and run the following commands. Note, you will need to add Heroku CLI to your command line.

```
>git add .
> git commit -m "your commit message"
>git push heroku master
```

This will likely take a few minutes to run. Basically, what it is doing is compiling and compressing your work into an optimized build and then deploying that to the Heroku server. Once this finishes you will see a build successful and it will say it is ready. Enter the following command to open your deployed app

```
>heroku open
```

4.2 Database Schema

In the database schema is illustrated in Figure 20. Tthere are 11 tables, 6 of which are the core tables and the remaining 5 are log tables. The "users" table is the table that stores a username, password and corresponding "physicianID" (foreign key). This is used for logging into the application. The "Physician" table has the primary key physician ID. The physicianID is the most widely used foreign key as it is passed around when routing between pages. The "Patient" table has primary key "patientID" and foreign key "primaryPhysicianID". The "Patient" table is the core table of this project. The "MedHistory" table has primary key "medHistID" and foreign key "patientID."The "SubscribedPatients" table has two foreign keys "physicianID" and "patientID." The "PatientEvent" table has primary key "patientEventID" and two foreign keys "patientID" and "eventCreatorID."

Most of the fields in the database schema are intentional and should explain what values they hold. It is worth noting however that the table PatientEvent used the foreign key eventCreatorID which is the physicianID.

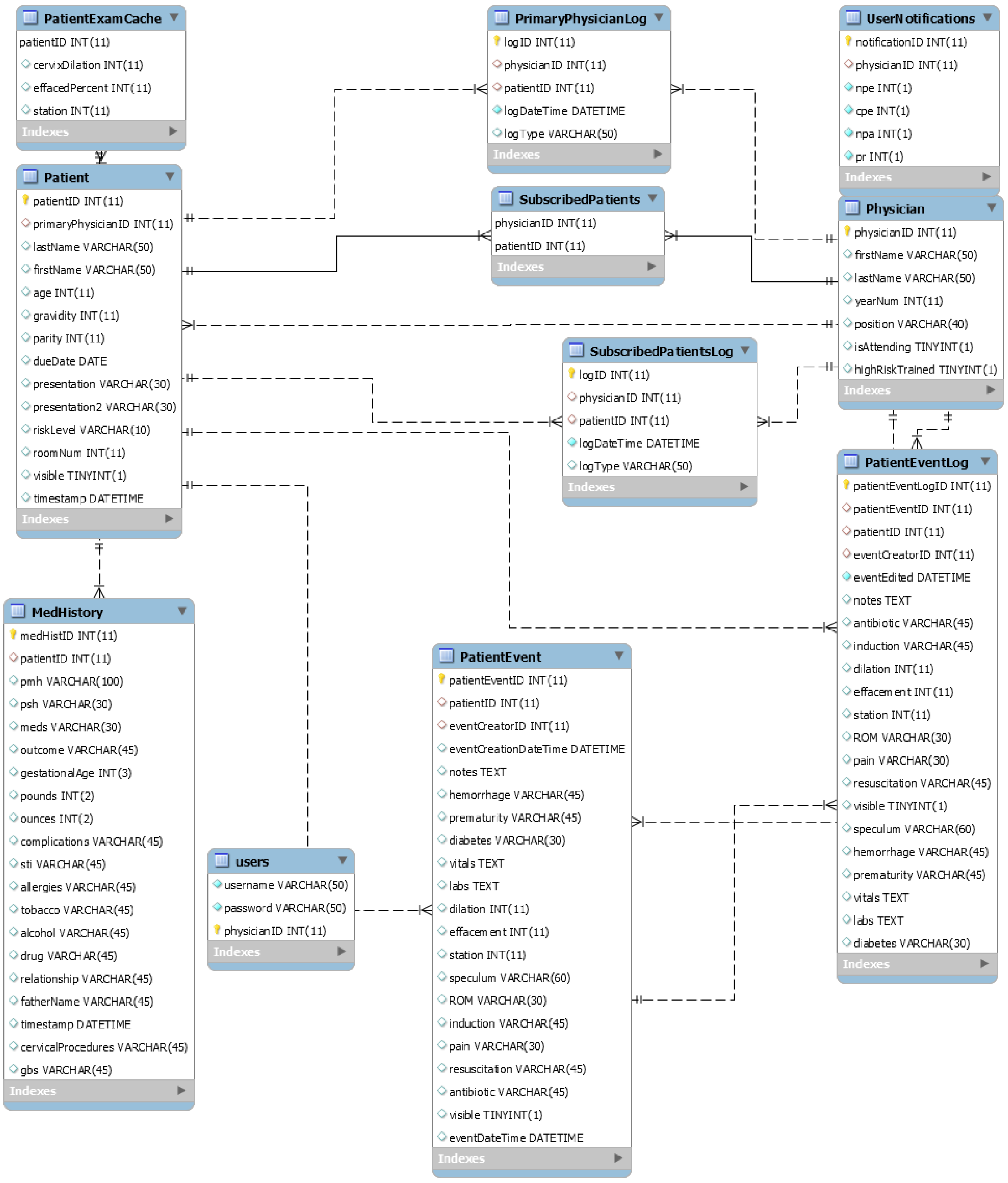


Figure 20 Database Schema

5. Recommendations and Conclusions

5.1 Future Features

The timeline for project did not lend itself to developing a fully developed application. There are still bugs and features that are desirable to implement. The recommendations for future features to implement are presented in Table 2. After these features are implemented, there are two further actions that should be taken. The first is a translation into React Native and then the implementation of platform notifications for subscribed patients. This will likely take longer than all of the features shown in Table 2 to implement and would be a great step in getting this application used by the hospital. The second step would be to have the application integrate with the EHR system so that the information captured within the application can be easily transferred to the EHR system. The app must also comply with FHIR [*]and HL7 standards. These are security standards that need to be in place before the application can be integrated into the hospital. When these two steps are finished the hospital only needs a way to deploy the application and then it can be integrated into the daily work of clinicians.

5.2 Future Deployment

Currently the application is in its infancy development stage. It is my recommendation to continue to iteratively develop the application using Heroku until a sufficient application is developed. At that point it would be best to try and move the application to another host. Previously the application was hosted on a dedicated UNIX server. This deployment has the advantage that we can securely run field tests with the application as the entire application is secure behind WPI servers. It does however induce stress issues when many users are using the application at the same time. My recommendation would be to reach out to network operations at WPI and try to come up with a strong host. In doing so the application will be hosted on an internal secure network and still be strong enough to withstand stress. Of course, since the end goal is to have the UMass Memorial Hospital to use the application it may also be worth seeing how they would want the application to be deployed. If a fully developed and secure app is created the hospital would not use a WPI server to run the application. Instead they would likely use a deployment service. In this case a conversation with how the app should be deployed should be had. It is more than likely the IT department of the hospital will be the ones to deploy the application but future workers on this project will need to be part of that process for it go run smoothly.

[*] https://www.hl7.org/fhir/

Table 2 Future Features

Feature	Priority	Challenge Level
Edit Patient Page	High	Easy
Edit Event Page	High	Easy
Edit Medical History Page	High	Easy
Remove Patient Button	High	Easy
Fully Functional NavBar	High	Medium
Hashing Password into Database	High	Easy-Medium
Fix Login Bug[†]	High	Hard
Whiteboard	High	Medium
Complete Med History Page	High	Hard
Subscribe/Unsub Slider Fixed	Medium	Easy
Style AllPatient-Patient Component	Medium	Medium
Template Exams[‡]	Medium	Hard
Style AddEvent	Medium	Easy
Style AddPatient	Medium	Easy
Style AddMedHistory	Medium	Easy
Add Main Page in App.js	Medium	Easy
Add GBS field to Table Patient	Medium	Easy
Add GBS field in AddPatient Form	Medium	Easy

[†] The bug I am referring to is if you enter an incorrect username or password and then a correct username or password you will get a "headers" error. This error comes about because the header for the response was already sent and when the new username and password is attempted it tries to reset the header. I did not know how to resolve this issue. See login component, Databaseinteract and server.js to resolve this issue.

[‡] Template exams are prefilled events that only show the fields relevant to that event. This was a strongly desired feature by the clinicians

Make DD fields into Multiple Selects[§]	Medium	Medium

5.3 Conclusions

The main goal of this project was to develop a web application that would serve as the foundation for future work to iteratively develop a production ready application to use in labor and delivery wards. Due to time constraints, this app was not tested in the field. However, this application in its final stage does provide the functionality that the previous application had but has far more potential in being developed and compiled into native platforms via React Native. It is my hope that this project gets picked up and future work will be done to extend the application until it is integrated in UMass Memorial Hospital.

[§] The fields that should be multiple are listed in the table of features by Dr. Amir Mehdizadeh in the Appendix

References

Buna, S. (2017). Yes, React is taking over front-end development. The question is why. Retrieved from https://medium.freecodecamp.org/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76

Curtis, R., Friedlander, S., Gelinas, A., Hammer, C., & Perullo, A. (2018). *Developing A Mobile App to Aid Communication in a Maternity Ward*. Retrieved from

Facebook. (2019a). create-react-app. Retrieved from https://github.com/facebook/create-react-app

Facebook. (2019b). Getting Started. Retrieved from https://reactjs.org/docs/getting-started.html

Facebook. (2019c). React Native. Retrieved from https://facebook.github.io/react-native/

Google. (2019). Quick Start. Retrieved from https://angular.io/guide/quickstart

Hall, M. (2019). Heroku Buildpack for create-react-app. Retrieved from https://elements.heroku.com/buildpacks/mars/create-react-app-buildpack

Lowe, H., Nawab, S., & Servidio, T. (2019). *Improving Labor and Delivery Tracking App*. Retrieved from

Moa, M. (2018). Setting up a Node.js Express Server for React. Retrieved from https://medium.com/@maison.moa/setting-up-an-express-backend-server-for-create-react-app-bc7620b20a61

Tarnowski, D. (2017). Angular vs React - the DEAL BREAKER. Retrieved from https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc

TechMagic. (2018). React vs Angular vs Vue.js - What to Choose in 2019? Retrieved from https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

Appendix

Table 3 Design For Experience Recommendations (Lowe et al., 2019)

User Type/Role	**New feature/What they want or prefer**	**Why they want it**
Residents	User change setting for theme preference (light vs. dark theme)	Simply to change theme based on preference
Attending, Residents	Colored bar subscription marker	Preferred way of seeing subscription to patient compared to other options suggested
OB/GYN	Condensed event timeline	For better viewing of information, and wasting less space
Residents, Attending, Nurses, OB/GYN	Condensed one-liner home page with standard format	Keep information viewing as concise and similar to current system as possible
Residents, Attending, Nurses, OB/GYN	Implementing drop-down menus and other custom UI elements based on information being entered	Much easier in some cases to have preset values to choose rather than always using free text to enter information

Table 4 Design For Context Adaptability Recommendations (Lowe et al., 2019)

User type/Role	New feature/What they want or prefer	Why they want it
Residents, Attending	Be able to select milestone events from patient timeline to see on home page	To be able to see most important history initially and view more if needed later
Resident, Attending, OB/GYN	Have separate tab or section to enter patient note in which whole screen is available for free text	Important for nearly all roles to have detailed patient notes readily available
Postpartum nurse	A "read" stamp of sorts to see if messages sent are seen by the reviewer	Know when a doctor reads a message sent to them
Postpartum nurses	End of shift care plan note sections	To easily view nursing care and post-delivery care plans
Postpartum nurse	Include labs pending and discharge events (discharges should be flagged and approved on the app)	To see labs that have been done to better determine future labs. Discharges should be approved before patient leaves hospital so those involved with patient are aware
Attending	Alert to all attendings in emergency cases	Cases in which one of more attending may need to be present or the assigned attending is in the OR and needs coverage
Attending	Add history of C sections, size of largest vaginal baby, G+P, gestational age	Important to have relevant patient history readily available to know how to better treat the patient in a more efficient manner
Nurse Practitioner	Add blood type, ruptured membrane, patient's presentation, last time eating/drinking, GTPAL, GBS (+/-), any penicillin sensitivity	Information for nurse practitioners to better treat the patient in a more efficient manner

Nurse Practitioner	Input for different types of breech events -Complete, Incomplete, Frank, Footling	To be aware of orientation of the baby and how to proceed accordingly
Nurse Practitioner, OB/GYN	Options list for high risk patients (why they are high risk). Add drop-down with; Diabetes, Bleeding disorder, Previous C section, Fetal Anomaly	In order to not only help distinguish between high and low risk patients but between high risk and other high risk for the sake of assigning the patient and how to properly treat them
OB/GYN	Add Drop-down (field would like be *Chief Complaint*). Options would be induction of labor (why-term, failure to grow), SROM, Laboring, Premature Labor, Abrupting	Be able to choose from a list of options why the patient is there and what is going on with them to give correct treatment
OB/GYN	Add fetal heart rate field: choose from category 1, 2, or 3 (alerts for category 2 and 3 situations)	In order to know how the baby is doing and whether the baby should be watched more closely, or intervention needed to improve heart rate
OB/GYN, Attending	Search bar to find specific people from labor and delivery and joint departments and be able to pull up contact info	If you would like to make contact with a certain person to ask questions or follow-up on a patient
Resident	Notification for new event/update to an event (pop-up or number system similar to new message or email on phone, possibly bold new updates)	Keep residents aware of new patient events and updates either during their shift or updates that happened when they were not working
Nurse	Have patient vitals incorporated into the app	To help with those that are not nurses to be aware of patient vitals and keeping track of any issues with vitals

Nurse	Have assignments sent from charge through the app so that assignments do not have to be written on paper and handed out	Improve efficiency with nursing assignments and cut down time for nurses meeting in nursing ed room with the charge nurse
Family medicine	Add gender/circumcision (baby), pain management of mother and list of labs	These are pieces of information that are commonly asked multiple times and it would be helpful to have it available on the app
Resident	Delete event button	In the case that an event is no longer pertinent or has been updated many times, or in cases of being entered incorrectly

Table 5 Design for Workflow Integration Recommendations (Lowe et al., 2019)

User Type/Role	**New feature/What they want or prefer**	**Why they want it**
Resident, Attending, Nurse, OB/GYN	Integration of the app with EPIC electronic medical records used at the hospital. Have information entered into the app be able to be shared with EPIC and allow the app to pull pertinent information from EPIC when possible.	For efficiency and ease of use. Be able to share and update information as efficiently as possible.

Table 6 Dr. Amir Mehdizadeh Terminology and Field Recommendations

One liner			
First Name		Free text	
Last Name		Free text	
OB/GYN		List of known attendings / clinics, free text for transfer from outside provider / hospital	
Age		Numbers 10 - 60	
Gravidity		Numbers 1 - 99	Number of times patient has been pregnant
Parity	Term	Number 0-99	Number of pregnancies >37 wks
	Preterm	Number 0-99	Number of pregnancies 20 - 37 wks
	Abortions	Number 0-99	Number of pregnancies 0-20wks
	Living	Number 0-99	Number of currently living children
GBS		Negative	Group B Strep presence / sensitivity
		Positive	
		Positive clinda / erytho sensitive	
		Positive clinda / erytho resistant	
Presentation - A		Vertex	Presentation of Baby A
		Breech	
		Transverse	
Preseentation - B		N/a	In case there are twins, presentation of baby B
		Vertex	
		Breech	
		Transverse	
Past Medical history			
Ob History			If gravidity = N, then the use should do the OB history for N-1 pregnancies
Outcome		SVD	
		LTCS	

		classical CS	
		SAB	
		SAB w/ D&C	
		TAB	
		TAB w/ D&C	
		Ectopic	
		VAVD for NRFHT	
		VAVD for mat. Exhaustion	
		FAVD for NRFHT	
		FAVD for mat. Exhaustion	
Gestational Age		Number 0 - 50	
Weight	lb	number 0-20	
	oz	number 0-15	
Complications		Denies	
		gDM	Can choose multiple
		PPH	
		gHTN	
		preE	
		preE w/ SF	
GYN Hx			
STI		Denies	Can choose multiple
		Gonorrhea	
		Chlamydia	
		Herpes	
		Syphilis	
		HIV	
		Trichomoniasis	
Cervical procedures		Denies	
		S/p LEEP	
		S/p CKC	
		S/p cryotherapy	
PMH		Denies	can choose multiple
		cHTN	
		Asthma	
		Hypothyroidism	
		DM1	

		DM2	
		Anxiety	
		Depression	
		Bipolar	
		GERD	
		HLD	
		free text	
PSH		Denies	Can choose multiple
		D&C	
		appy (lsc)	Pull in Csections and number from OB hx
		appy (open)	
		chole (lsc)	
		chole (open)	
		myomectomy (lsc)	
		myomectomy (open)	
		myomectomy (hysteroscopic)	
		WT	
		T&A	
		salpingectomy (lsc)	
		salpingectomy (open)	
		cystectomy (lsc)	
		cystectomy (open)	
		free text	
Meds		Denies	Can choose multiple
		PNV	*need way to include dose / frequency ?free text
		iron	
		zofran	
		tums	
		ranitidine	
		sertraline	
		levothyroxine	
		labetalol	
		nifedipine	
		methyldopa	
		humalog	
		novalog	
		regular insulin	

		NPH	
		Lantus	
		Levemir	
		free text	
Allergies		NKDA	free text should have option for med / reaction
		free text	
Social Hx			
Tobacco		Never smoker	
		Former Smoker	
		Current smoker	
		free text	
Alcohol		Denies	
		Social alcohol	
		free text	
Drug		marijuana	can choose multiple
		cocaine	
		heroin	
		prescription opioids	
		benzodiazepines	
		barbituates	
		meth	
		ecstasy	
		LSD	
FOB	Name	*free text*	
	Relationship	Husband	
		Fiance	
		Boy friend	
		Sperm Donor	
		No relationship	
Other		*free text*	
Problems		None	can choose multiple
		pior c/s	
		cHTN	
		gHTN	
		preE w/o SF	
		preE w/ SF	

		gDM	
		DM1	
		DM2	
		AMA	
		fetal anomaly	
		IUGR	
		macrosomia	
		mo-mo twins	
		mo-di twins	
		di-di twins	
		multiples (other)	
		IVF	
		free text	
Exam	Dilation	0,0.5,1,1.5 … 10	0-10 in 0.5 intervals
	Efacement	0,5,10,15 … 100	0-100 in 5 intervals
	Station	"-3, -2.5, -2, -1.5, -1, -0.5, 0, +0.5"	"-3 to +3 in 0.5 intervals"
	Speculum	pool positive	Can choose multiple
		pool negative	
		fern positive	
		fern negative	
		visually closed	
		visually dilated	
		no blood	
		scan old blood	
		active bleeding	
		herpes negative	
		herpes positive	
Induction		Misoprostol PV	Can choose multiple
		Misoprostol buccal	
		Cervidil	
		Cervidil out	
		Pitocin	
		Pitocin reduced	
		Pitocin off	

		Cooks foley	
		Cervical foley	
		Foley out	
Membranes		SROM clear	Can not choose multiple
		SROM meconium	
		AROM clear	
		AROM meconium	
Maternal ressucitation		IVF	Can choose multiple
		O2	
		IUPC	
		Amnioinfusion	
		Terbutaline	
Pain		Nubain	Can choose multiple
		Morphine	
		Fentanyl	
		Epidural	
		Tylenol	
		Fioricet	
		PCA	
Antibiotics		Penicilin	Can choose multiple
		Ampicilin	
		Cefazolin	
		Clindamycin	
		Vancomycin	
		Gentamicin	
		Unasyn	
Post Partum Hemorrhage		Misoprostol	Can choose multiple
		Methergine	
		Pitocin	
		Hemabate	
		Transexmic acid	
		Barki balloon	
Hypertnesion		Labetalol	Can choose multiple
		Nifedpine	
		Hydralazine	
		Magnesium	

		Valium	
		Calcium gluconate	
Prematurity		Betamethasone	Can choose multiple
		Nifedpine	
		Indomethacin	
		Magnesium	
Diabetes		Novalog	
		Humalong	
		NPH	
		Insulin drip	
Vitals		free text	
Labs		free text	