Dublin Business School

Higher Diploma in Science in Computing

(Software Development)

Final Report

Solution Title: thÉireP || ePrescriptions

Student Details:

Cian Walker

10021419@mydbs.ie

Student No: 10021419

Project Supervisor: Dr. Obinna Izima

Submission Date: 06/02/2023

[GitHub Repository Link](https://github.com/Cian0o/HDipThesis)

**Abstract:**

Having built a basic Python/Flask/SQLite CRUD application for ePrescriptions as part of my final module in Advanced Programming ([GitHub Repository here](https://github.com/Cian0o/AdvProgCA)), I wished to deepen my knowledge and skills in full-stack web-application development.

In particular: I wished to emphasise learning in both user authentication and the latest and most relevant development technologies used in the wild today.

As is attested to within the proposal and even within the interim report, I had only a couple of tooling certainties decided upon until quite a late stage in the project lifecycle. I knew I wanted to implement the application in JavaScript and to build it in a Node.js runtime environment. Aside from that, I envisaged serving HTML from a single backend server in much the same way as a Python/Flask/SQLite stack would, albeit with much more dynamic elements controlled by more JavaScript.

It was apparent to me early on that Node.js is used in conjunction with the Express server framework and is increasingly being back-ended by a MongoDB non-relational database. What I stumbled upon quite late – late November, with only approximately one third of the project timeframe remaining – was the React.js frontend framework and the MERN stack that it rounds out in conjunction with the technologies already mentioned above.

With the retrospective knowledge that a React frontend framework is multiple times more complex that the rest of an application stack combined, it was foolhardy on one level to attempt to implement this layer with only one of the three months remaining. Some of the functionalities and interfaces are rough and patchy as a result. On another level, using the MERN stack and gaining an understanding of how these technologies integrate has been a profoundly useful learning experience. The insights and experience I’ve been forced to glean as regards authentication in addition to application state and properties have been invaluable.

**Acknowledgments:**

Further to previous acknowledgements in my interim report to Robert Graham and Gonzalo Lorca of IBM, my supervisor Dr. Izima, Claire Caulfield, Jennie Byrne, and Jon Harry: I’d like to especially acknowledge Bob Kalka, global head of Security PreSales Engineering, and Constantinos Stavrou, Security PreSales Engineering lead UK & Ireland of IBM, for facilitating and encouraging my transition to a new technical direction in life. My partner, Gillian Murphy, also kindly and patiently endured endless recitation of the logic behind MERN-stack development so that I could get it straight in my own head. Her generosity took a particular sting out of the process for me.

**Contents**

List of contents, including tables and figures, with page numbers.

**Chapter 1: Introduction**

By far the greatest additions and contrasts to be found in this report since its interim counterpart are all related to the React.js frontend framework and the learning paths it has led me down. What I had learned virtually nothing of when building my Python/Flask/SQLite stack was the concept of application state and properties. My learning steered itself in this direction quite inevitably as a consequence of my interest in identity & access management. While user authentication is far from the only aspect of application state, it is an elemental one. The available literature, training tutorials, and materials out there remain quite disjointed and dependent on almost infinite different combinations of technologies. The MERN stack seemed the combination most likely to have consistent and reliable pedagogical materials available to it, in addition to being extremely ubiquitous, current, and relevant. Coming around to the technical meat and bones of the implementation, I would state that the greatest change in my perspective since the interim report is the following. Barely six weeks ago, I conceived of front-end and user interface as synonyms for the same material object. I now realise that user interface is but a subset of front-end development. Application state and properties and their management can be a gargantuan undertaking, and one that is intimately intertwined with the discipline in which I am an apprentice engineer: Identity & Access management. Following the implementation of my Python/Flask/SQLite stack, I’d imagined a web application as being composed of three layers:

* HTML styled with CSS and dynamicised by JavaScript.
* A middle layer of REST APIs performing HTTP crud functions between HTML pages and a database.
* A relational or non-relational database being read, written to, updated, and deleted.

Reminiscent of how time is thought of as a fourth dimension, I now realise the importance of the consideration of *statefulness*, which I understand to mean the status of a user’s session within the application. To reiterate: the combination of motivation to gain skills in identity and access, coupled with the need to choose a technology that had ample learning resources available to it, plus the fact that I had already determined to implement the platform in a Node.js runtime environment, meant that a MERN stack web application was the only realistic direction of travel at that time. Statefulness aside, it was important to learn how to design and implement a single page web application as per the expected industry standard today. React.js, along with Angular.js and Vue.js, is one of the three pre-eminent frameworks in this regard also.

**Chapter 2: Background/Literature Review**

Around the time of my completion of the Python/Flask/SQLite stack, I had a mentoring session with an IBM colleague, Robert Graham. Robert delivers large Identity & Access projects with our clients post-sale and recommended that I become familiar with Node.js given my particular interest in consumer-facing identity and access, and because that is increasingly what developers are implementing.

As I delved into the JavaScript language generally, I came to realise that it had evolved into something completely unique. I would describe it as having evolved from the user interface — where it was initially just a markup element orchestrator and dynamiciser (Eich, 2011) — back ‘down the valve’ (for want of a better term) into the server where it came to be used to write APIs. This seems singularly unique for a programming language. Further to this, JavaScript Object Notation (JSON) had become arguably the most common non-SQL data format in use. I began to notice the emerging ‘*JavaScript Everywhere’* (Rauch, 2012) trend and became keen to write my web application from one end to the other in one programming language. Furthermore, learning JavaScript is somewhat unique compared to other languages in that it’s originally designed to orchestrate and renders user interface elements. Where development students may spend a lot of time playing around with Python or C-type languages in a command line-style interface, JavaScript feels ‘real’ very quickly as youcan quickly and easily run your code in a browser. I’ve somewhat fallen in love with it as a language. I hope I will receive some credit for being almost completely self-taught in this regard as it was only covered as an as aside and as a fraction of a topic in two modules.

Beyond JavaScript as a mere language, it quickly became apparent that Node.js was a very suitable programming paradigm to use for an application scoped the way *thÉireP* is. In an application where there is minimal processing of data sets, Node’s non-blocking in/out event loop is optimised for speed of HTTP request and response (Chitra & Satapathy, 2017). This asynchronicity is a textbook ‘hack’ that sweats performance from assets without requiring increased compute power or indeed even mbps network performance:

*“As an example, let's consider a case where each request to a web server takes 50ms to complete and 45ms of that 50ms is database I/O that can be done asynchronously. Choosing non-blocking asynchronous operations frees up that 45ms per request to handle other requests. This is a significant difference in capacity just by choosing to use non-blocking methods instead of blocking methods.” – Node.js Documentation.*

In light of this, I was positive that Node.js was an ideal programming paradigm in which to wite an effective application for registering and authenticating users; and submitting, retrieving, updating, and deleting prescription records.

By the point in time of late November to early December, I had become quite familiar with JavaScript and had built some nice dynamic user interface but was still struggling on the research front in terms of finding frameworks and tutorials for integrating user authentication workflows into a web application. I was particularly adamant of my need to learn to implement an OAuth/OpenID Connect-style authentication flow that uses JSON Web Tokens (JWT) in browser local storage to determine if a user is authenticated or not. I’ve understood this principle for quite some time now being an Identity & Access solutions engineer but I was extremely anxious to learn the nuts and bolts of *how* this is implemented.

**Chapter 3: Specification and Design**

Design of the artefact should be discussed in this chapter which is augmented by diagrams.

Being honoestt – didn’t know what I was getting in to / how big it’d be

**Chapter 4: Implementation**

The student should discuss the implementation details here. It includes software and version of software used. Any algorithm developed during this process may be discussed. Any coding snippets can be discussed. *Please note, full code should be included in appendix not in this*

*chapter.*

**Chapter 5: Testing and Results**

This chapter contains a clear description of testing and the results of the project. The chapter also contains a description of the results of the final tests carried out on the product.

An important section in this chapter is the *critical evaluation* of the final project, where the student demonstrates the ability to critically evaluate the work done, the shortcomings in the project and so on. Objectivity is important when writing this chapter.

**Chapter 6: Conclusions and Future Work**

A review of what the project achieved, a final review of the project in terms of the proposed goals and project plan.

● Any changes from the interim report should be discussed and justified.

● The student should reflect on the learning experiences gained in doing the project and its relevance to on-going progress as a learner and future practising IT professional.

● This section should also provide a starting point for another student to continue the work.

**References and Bibliography**

Harvard referencing style.

**Appendices**

Note: Well-documented code listings should be included in an appendix, not in the main body of the report.

PROPS = PROPERTIES!

: <https://expressjs.com/>

<https://www.pagecloud.com/blog/how-to-add-custom-fonts-to-any-website>

<https://techtrim.tech/express-vs-flask/>

<https://www.w3schools.com/css/css3_buttons.asp>

<https://www.youtube.com/watch?v=Y_yw5L-D3IQ&ab_channel=RizwanKhan>

Use DOM in Node: <https://github.com/jsdom/jsdom>

<https://codingsans.com/blog/node-config-best-practices>

To Do:

Drop-Down Login & Logo with Coloration

<https://stackoverflow.com/questions/28863097/template-inheritance-with-node-js-handlebars-and-express>

<https://www.youtube.com/watch?v=WqMbzVWIAjY>

PATCH not PUT for amending prescription!

Surgeries should have prescribe (POST), amend (PATCH), and Delete (plus autodelete when expired)

Pharmas should have retrieve (GET) and mark prescription as dispensed?!?

Prescribe, Register both Doc & Pharmas forms with rendering of record into UI ( <div id=”output”> )with confirm buttons and clear buttons w/iterators (or uuid for random IDs?)

Put: Updating records ‘Working with thee edit state’?

New JSON entity shemas?

Form validations – REGEX?

PSI & MCI Nos validator?

Disappearing / Timeouts of warning/error notices

Push and pull pharma, doctor regs , and prescription JSONs to and from local storage

Easy HTTP library API Calls? / fetch API?

Arrow functions\*

<https://kinsta.com/blog/nodejs-vs-python/>

CSS Grid: [https://en.wikipedia.org/wiki/CSS\_grid\_layout /](https://en.wikipedia.org/wiki/CSS_grid_layout%20/) <https://www.youtube.com/watch?v=0xMQfnTU6oo>

MongoDB: <https://www.youtube.com/watch?v=2QQGWYe7IDU>

Bibliography:

Abeln , M. (2019) *What is the purpose of having two running ports when we working with reactjs and nodejs?*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/57362813/what-is-the-purpose-of-having-two-running-ports-when-we-working-with-reactjs-and (Accessed: December 19, 2022).

Admin (2022) *Redux in ReactJS*, *W3schools*. Available at: https://www.w3schools.blog/redux-reactjs (Accessed: December 20, 2022).

Bosler, F. (2020) *Set up an express.Js app with passport.js and mongodb for password authentication*, *Medium*. The Startup. Available at: https://medium.com/swlh/set-up-an-express-js-app-with-passport-js-and-mongodb-for-password-authentication-6ea05d95335c#94df (Accessed: January 4, 2023).

Chitra, L.P. and Satapathy, R. (2017) “Performance comparison and evaluation of node.js and traditional web server (IIS),” *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)* [Preprint]. Available at: https://doi.org/10.1109/icammaet.2017.8186633.

Codes, C. (2022) *Google Identity Services Login with react (2022 react Google Login)*, *YouTube*. YouTube. Available at: https://www.youtube.com/watch?v=roxC8SMs7HU&t=944s&ab\_channel=CooperCodes (Accessed: January 5, 2023).

*Conditional Rendering* (no date) *ReactJS*. Available at: https://reactjs.org/docs/conditional-rendering.html#gatsby-focus-wrapper (Accessed: January 3, 2023).

*CORS* (no date) *Node Package Manager*. Available at: https://www.npmjs.com/package/cors (Accessed: December 22, 2022).

Crockford , D. (2011) *Douglas Crockford: The JSON Saga*, *YouTube*. YUI Library. Available at: https://www.youtube.com/watch?v=-C-JoyNuQJs&ab\_channel=YUILibrary (Accessed: December 6, 2023).

Eich, B. and Friedman, L. (2021) “Brendan Eich: JavaScript, Firefox, Mozilla, and Brave | Lex Fridman Podcast #160,” *The Lex Friedman Podcast*. Available at: https://www.youtube.com/watch?v=krB0enBeSiE&ab\_channel=LexFridman (Accessed: November 30, 2022).

Erikson, M. (2022) *Redux createstore() is deprecated - cannot get state from getState() in Redux Action*, *Stack Overflow*. Available at: https://stackoverflow.com/a/71947129/19219155 (Accessed: December 20, 2022).

Fatunmbi, T. (2022) *A comparison of cookies and tokens for secure authentication*, *Okta Developer*. Okta Inc. Available at: https://developer.okta.com/blog/2022/02/08/cookies-vs-tokens#:~:text=Cookies%20and%20tokens%20are%20two,characters%20created%20by%20the%20server. (Accessed: October 4th, 2022).

*JavaScript Async* (no date) *W3 Schools*. Available at: https://www.w3schools.com/js/js\_async.asp#:~:text=async%20makes%20a%20function%20return,function%20wait%20for%20a%20Promise (Accessed: October 19, 2022).

*JSX in depth* (no date) *React*. ReactJS. Available at: https://reactjs.org/docs/jsx-in-depth.htm (Accessed: December 13, 2022).

\*\*On accepting passwords at login\*\*:

Maaajomaaajo 74955 silver badges1010 bronze badges, K.E. (2020) *What is the Difference Between handlechange vs onchange in which is used in react?*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/62197917/what-is-the-difference-between-handlechange-vs-onchange-in-which-is-used-in-reac (Accessed: December 31, 2022).

Node.js (no date) *Overview of blocking vs Non-Blocking*, *Node.js*. Node.js. Available at: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/ (Accessed: December 13, 2022).

Olawanle, J. (2022) *Post HTTP Request in React*, *Stack Abuse*. Stack Abuse. Available at: https://stackabuse.com/post-http-request-in-react/ (Accessed: December 21, 2022).

*Openid Connect* (2022) *IBM.com*. International Business Machines Inc. Available at: https://www.ibm.com/docs/en/was-liberty/base?topic=liberty-openid-connect (Accessed: December 19, 2022).

Patadiya, J. (2022) *React vs react native - key difference, features, advantages*, *Radixweb*. Radixweb. Available at: https://radixweb.com/blog/react-vs-react-native (Accessed: December 19, 2022).

Raj, V. (2019) *Running react and node.js in one shot with concurrently!*, *DEV Community*. DEV Community. Available at: https://dev.to/numtostr/running-react-and-node-js-in-one-shot-with-concurrently-2oac (Accessed: December 19, 2022).

Rauch, G. (2012) *Smashing node javascript everywhere*. New York: Wiley.

*React ES6 arrow functions* (no date) *W3 Schools*. Available at: https://www.w3schools.com/react/react\_es6\_arrow.asp (Accessed: December 19, 2022).

*React ES6 spread operator* (no date) *W3 Schools*. Available at: https://www.w3schools.com/react/react\_es6\_spread.asp (Accessed: December 19, 2022).

Sakimura, N. *et al.* (2022) *OpenID Connect Core 1.0 incorporating errata set 1*, *OpenID Connect Core 1.0* . OpenID Connect. Available at: https://openid.net/specs/openid-connect-core-1\_0.html (Accessed: December 22, 2022).

*Start using your openid* (2012) *OpenID*. Available at: https://openid.net/start-using-your-openid/ (Accessed: November 8, 2022).

*Throw, and Try...Catch...Finally* (no date) *JavaScript Errors*. W3 Schools. Available at: https://www.w3schools.com/js/js\_errors.asp#:~:text=JavaScript%20try%20and%20catch,occurs%20in%20the%20try%20block. (Accessed: November 3, 2023).

Tiwari, V. (2022) *How to fix cors error "no 'access-control-allow-origin' header is present on the requested resource"?*, *codedamn*. Available at: https://codedamn.com/news/backend/how-to-fix-cors-error (Accessed: December 22, 2022).

Traversy, B (2019) *DevConnector\_2.0: Social Network for Developers, Built on The MERN stack*, *GitHub*. GitHub. Available at: https://github.com/bradtraversy/devconnector\_2.0 (Accessed: November 26, 2022).

*Understanding and Resolving Cors Error* (no date) *Contentstack RSS*. Available at: https://www.contentstack.com/docs/developers/how-to-guides/understanding-and-resolving-cors-error/ (Accessed: December 22, 2022).

*Using the Effect Hook* (no date) *ReactJS*. Available at: https://reactjs.org/docs/hooks-effect.html (Accessed: December 24, 2022).

*Using the State Hook* (no date) *ReactJS*. Available at: https://reactjs.org/docs/hooks-state.html (Accessed: December 19, 2022).

*Why redux toolkit is how to use Redux Today* (2022) *ReduxJS*. Available at: https://redux.js.org/introduction/why-rtk-is-redux-today (Accessed: December 20, 2022).