

PPIT Project 2021

## Buy & Sell Vehicles Website

Kealan O'Callaghan – Dean McGowan

Supervisor - Martin Kenirons

## **Work Responsibilities**

Mongo Database Setup – Kealan

Server Code / Schemas – Kealan/Dean

Login/Register/Jwt Setup – Kealan

Shop Add product page – Dean

Shop Cart – Dean

Vehicles Display page – Dean/Kealan

Document – Kealan/Dean

NavBar/Front end – Dean

Reading Writing from database - Kealan

## Introduction

Our project is a buy and sell web application specifically for vehicles. We aim to provide a clean and user-friendly front end that anyone can use. The application will allow users to register an account and then use that account each time they log in. Users can also post their own advertisements to the store. The user will also be able to add vehicles to a cart to be purchased. This product is aimed more so at car dealerships, who may have more than one of the same vehicle for sale.

## Technology Used

We decided to go with a mern-stack style, react application for a few reasons. We have had some experience with this technology before in our data representation module, but also, the prevalence of React in modern web development is huge. We thought it would be a great idea to do something that would stand to us in the future when applying for jobs in industry. Many companies are shifting towards react because of its ease of use so it was an easy choice for us as to which technology we wanted to use.

We used a wide array of technologies for this application. We used 'Node Js' for the server, a 'Mongo DB' database to store both user and product information, 'Nodemon' for the server updates, 'JWT' for the token authentication and finally 'React' for the application build.

Using **Node.js** had a multitude of benefits, to list a few:

### ***Robust Technology Stack***

Using Node.js for backend, we automatically get all the pros of full stack JavaScript development, such as better efficiency and overall productivity, code sharing and reuse, speed and performance, easy knowledge sharing with each other, and a huge number of free tools. We had better efficiency and overall developer productivity. This made the process a lot more flexible, the development is less time-consuming and as a result, we got fast and reliable software. With the same language on both sides, we were able to reuse code on the frontend and the backend by wrapping it into modules and creating new levels of abstraction.

### ***Fast Processing and Event Based Model***

Node.js is certainly fast, the V8 engine utilised in Node.js implementation was originally developed for the Chrome browser. It is used to compile functions written in JavaScript into machine code, and it does this at a great speed.

Another aspect is the event-based model. When using a common language for both client/server-side, synchronization happens fast, which is especially helpful for event-based, real-time applications like ours.

### ***A Rich Ecosystem***

The default Node.js package manager, npm, is an incredible marketplace for open source JavaScript tool.

**React** will be used as the main building block of the application for many reasons. To list a few:

### ***Rich User Interfaces***

The quality of the user interface in an application plays an important role. If it is poorly designed, then it lowers the chances of an application to succeed. React allows the building of high-quality, rich user interfaces through its declarative components.

### ***Fast Rendering***

React will allow the user to update and receive information without reloading pages. The use of the virtual DOM also allows testing changes first to calculate risks with each modification. This maintains high performance.

**Redux** was used in the login/register for global state management. We used Redux because it simplifies storing and managing component states. Redux stores application state in a single object and allows every component to access application state.

**Json** web token will be used as the token authenticator as its self-contained and contains all the information it needs for authentication.

### **Database**

For our database, we chose MongoDB. We chose MongoDB as it is very well suited to the style of data, we would be using e.g. Name value pairs, which we are familiar with.

It is also very well paired with node.js and we knew if we ran into any complications that there would be many resources to help us troubleshoot them.

### **Schema:**

There are two collections in our mongo database, one for the users of the website, which includes their name, email address and password for logging into the website, and another collection to store each of the products which includes, name of the product, the price, a short description, quantity in stock and an image url to be displayed for the product.

### Architecture of the application

The user is first met with a 'Register' or 'Log In' screen. If the user is unregistered, they can do so by clicking the 'Register' button and supplying details. They can then click 'Log in' and be brought to the products page. If a user has already registered and logged in on the device previously, they will be brought straight to the products page after clicking 'Log in' without the need of supplying their details.

**Welcome** to Kealan & Dean's Project

REGISTER

LOG IN

The register page is a form which asks the user for a name, email, password, and to confirm the supplied password. If the user arrives at the register page but has already made an account, there is a 'Log in' button to bring you to the log in page.

[← BACK TO HOME](#)

**Register** below

Already have an account? [Log in](#)

Name

---

Email

---

Password

---

Confirm Password

---

SIGN UP

A Navbar that displays the name of the application and links to various pages such as: For Sale, Post Ad, and Cart

**Buy&Sell.com**

[For Sale](#)



[Post Ad](#)

[Cart](#)

0

The 'For Sale' page is a collection of card like advertisements for vehicles. All these items are stored in the database. It gives information like brand, model,

year, a short description, and an image. It also has an 'Add to Cart' button.

 <b>Toyota Corolla 2004</b> €800 Reliable Car 1 Available <a href="#">Add to Cart</a>	 <b>Bmw 5-Series 2014</b> €13350 Like new, 10k miles on each 2 Available <a href="#">Add to Cart</a>
---	--

The 'Post Ad' page is a form that allows the user to give a product name, price, stock availability, short description, and a link to an image representing the item. The user then uses the submit button to add this information to the 'For Sale' page.

**Product Name:**

---

**Price:**

---

**Available in Stock:**

---

**Short Description:**

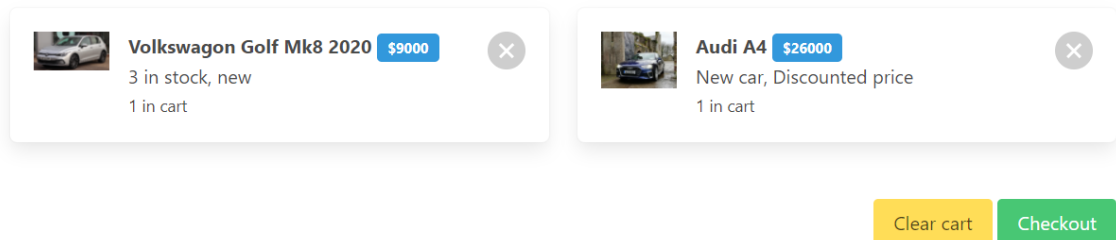
---

**Image:**

[Submit](#)



The 'Cart' is another card like structure which shows all items currently added to the cart. Each item can be selectively deleted by using the 'X' button at the top corner of the card. The 'Clear Cart' button will remove all items currently in the cart. The checkout button currently has no functionality.



## **Features of The Project**

A few of the key features for the project include the following:

- A multi-user system
- A login/register page that utilises token authentication
- Talking to a Mongo Database to store user credentials and vehicle information
- Reading from a Mongo Database to populate the 'For Sale' page.

## **Implementation of Features**

We started with the log in/register page. It is a minimal full-stack login using the MERN stack. It also uses Mongo DB for the database, Express and Node for the backend, and React for the frontend. We also integrated Redux for state management for our React components.

Firstly, we created the backend. We initialised our backend using npm and installed necessary packages. We then set up a MongoDB database. We used Node JS and Express to set up the server. We created a database schema to define a user for registration and login purposes. We then set two API routes (register and login), using passport and jsonwebtokens for authentication and validator for input validation.

We then began the frontend using create-react-app. We made static components for our Navbar, Landing, Login, and Register pages. We also set up redux for global state management.

Finally, we connected the backend with the frontend. We linked Redux to our components. We also added code that would let the user stay logged in whenever the page was refreshed or if they leave the page altogether.

For the Buy and Sell, we used React to scaffold out the interface of a basic shopping cart, products page, and add product page. We used context to move data and methods between multiple components. We then got the application reading the data from our MongoDB database so that the products remained on the app after you left.

## **Limitations**

During our project we knew there would be some limitations as we knew in advance, we were not going to be able to develop a full stack, final version application ready for industry. Some of these limitations were:

- The checkout functionality – at the time of development we didn't think it was feasible to include a proper visa or PayPal checkout as the products advertised on the website so far are just for presentational purposes.
- Advertising other products – for the sake of continuity and time management, we thought it would be best for us to stick to selling one product (vehicles) so that the website wouldn't get too cluttered. But in future development we think it would be a good idea to have a separate section where people could post anything for sale while the vehicles section would have its own page.

## **Complications and learning outcomes**

- 1<sup>st</sup> March: There was an issue with the jwt as it couldn't get a handle on the token and was returning 'undefined' meaning the user couldn't login to the landing page. This was later solved by making sure all the classes could access each other with the exports to give the classes a handle on their variables and tokens.
- 23<sup>rd</sup> March: We were able to get the products adding to the app, but it wasn't connecting to MongoDB, so the products only stayed on the dashboard for as long as the session was going.
- 5<sup>th</sup> April: We managed to get the products connect to our MongoDB database so we could add products, but the app was unable to read the existing products already in the database.
- 7<sup>th</sup> April: We added a delete button to each vehicle product. However, upon further discussion we removed it as although it worked, it didn't fit with the products page as any user could delete another user's product.

Overall, we would say that our biggest learning outcome throughout the whole project was the ability to troubleshoot issues and complications by ourselves without turning to lecturers or peers for help. We also learned the importance of keeping everything extremely organized as when developing a full stack application like this it could be very easy to lose track of files and exports.

We also learned some extremely valuable lessons about teamwork during this project. We learned the importance of keeping your team mates updated with the progress of the project, working together to troubleshoot issues and also the importance of using software like github and Microsoft teams to ensure that the project is at its most recent version at all times.

### **Development Lifecycle**

Feb 13<sup>th</sup> – We created the react app and installed all necessary packages.

Feb 15<sup>th</sup> – Began Creating the login and register components to work by themselves.

Feb 17<sup>th</sup> – Connected Login and register to MongoDB to store the users.

Feb 18<sup>th</sup> – Login and register is fully complete, and we began the shop front end.

Mar 1<sup>st</sup> – Front end for the shop is complete but not connected to database yet.

Mar 7<sup>th</sup> – Added functionality to shop and cart, login is now connected to shop.

Mar 23<sup>rd</sup> – Post ad page finished but not saving to products page yet.

Apr 5<sup>th</sup> – Posted ads writing to the database but are not being read when refreshed.

Apr 6<sup>th</sup> – App reading from database.

Apr 7<sup>th</sup> – Delete button added, doesn't delete from db yet.

Apr 9<sup>th</sup> – removed delete button due to bug.

Apr 13<sup>th</sup> – Changed product description to image URL to be displayed on product page.

Apr 16<sup>th</sup> – Changed theme of website. Application is complete.

### **Future Development**

If we were to take this application further, I think we could add some exciting new features:

- The checkout functionality is at the top of that list. At the moment we have a button to check out, but we have not implemented any features for it.
- We would add separate pages for each product, with more information on each product, and maybe some information on how to contact the seller.
- We would add ID on each product so that if you log in, only you can delete the ads that you have posted and no one else.
- In the later stages of development, we would add a way of contacting the seller in the app with a text chat functionality between buyer and seller.

### **References:**

*Build an e-commerce website with react:*

<https://www.youtube.com/watch?v=wPQ1-33teR4>

*Understanding JWT Authentication with Node.js:*

<https://www.simplilearn.com/tutorials/nodejs-tutorial/jwt-authentication>

*Authentication and Authorization using JWT with Node.js:*

<https://betterprogramming.pub/authentication-and-authorization-using-jwt-with-node-js-4099b2e6ca1f>

Ecommerce site in react:

<https://www.sitepoint.com/how-to-create-an-ecommerce-site-with-react//>