# Intro

The aim of this project was to build an application that would interact with a Tier 3 architecture system and help a team develop and establish their DevOps capabilities and operations. The application I have worked on aimed to be a website that would store and display personal notes, reminders, images, checklists etc in one place. The theme it was based on was a ‘family fridge’ and I chose to develop it using Mongo DB, Express, Vue, and Node with a couple of additional tools to help.

I will outline the tools used in the “Tools Used” section and will then go on to explain the choices and where I used them in the “Implementation” section - which is a walkthrough of the progress I have made.

# Tools Used

### Libraries/Frameworks

* Vue
* Bootstrap
* Mongoose
* Express
* Node.js
* Babel
* Lint
* Nodemon
* Body-parse
* Bcrypt
* Cors
* AWK-SDK
* SweetAlert
* JWT

### Languages

* HTML
* CSS/SCSS
* Javascript
* Other
  + .JSON
  + .YML

### Cloud Tools

* Mongo DB
* Mongo DB Altas
* AWS S3 Bucket
* Azure Virtual Machine
* GitHub

### Software

* VS Code
* Postman
* CurrPorts
* GitHub Desktop
* Chrome
* Docker
* Git Bash

# Implementation

A screenshot of a computer

Description automatically generated

Figure 1: Configuring the VSCode terminal to point to Git Bash

A screenshot of a computer

Description automatically generated

Figure 2: Configuring the VSCode terminal to point to Git Bash

I started out by installing the tools I knew I would need in advance as this would take time to set everything up. Firstly, I installed the IDE/Editor I was going to use and downloaded some extensions for Vue.js to make coding easier. VSCode is one of my favourite editors as it is simple and smooth to use. It also has Git features built in and great support for Node.js development with JavaScript and other web technologies. ( *Code.visualstudio.com. 2021. Why Visual Studio Code?. [online] Available at: <https://code.visualstudio.com/docs/editor/whyvscode> [Accessed 22 December 2021].)* I know that Visual Studios is an IDE and is a much more powerful tool but I am much more at ease using VSCode, especially after using it for my year on placement. In addition, the application I am building uses web based languages so I can debug on the browser instead. I also then configured VSCode to use GitBash in the terminal as this is my preferred one as well. *(Figure 1) (Figure 2)*

Next I installed Postman as unfortunately the online Postman tool does not work with Localhost and since that is where most of my time developing will be spent, I need a tool that will help me test the API connections.

A screenshot of a computer

Description automatically generated

Figure 3: Creating and running a Vue.js application

A picture containing text

Description automatically generated

Figure 4: Testing a Vue.js application on the local 8080 port

After setting up the editor I am using I then went on to install node.js and create a simple Vue.js application using the Vue CLI. I find this is a good way to start a project as it builds the initial components for you. Once done I quickly fired up the Vue app using “npm run serve” and tested it on port 8080 just to check that it was all running okay *(Figure 3) (Figure 4).*

Text

Description automatically generated

Figure 5: Creating the backend connection to a server

Graphical user interface, application, Word

Description automatically generated

Figure 6: Testing the backend connection to a server

Once the frontend was initialised, I needed to set up the backend. A popular web stack is MERN which uses Mongo DB, Express, React and Node.js. Due to it’s popularity I thought it would be a good web stack to take inspiration from as there would be plenty of documentation to reference and learn from, as well as the comfort of knowing that it is less likely to become obsolete any time soon. React, Vue, and Angular are all JavaScript frameworks/libraries and can be used somewhat interchangeably and therefore I am using Vue instead of React.js for the web stack. This is for two reasons, the first being that I am more familiar with Vue.js, the second is that it is said that React requires more complex implementations for simple tasks in comparison to Vue. *(Medium. 2021. React vs Vue: Is Vue.js going to take over React in 2020?. [online] Available at: <https://medium.com/swlh/is-vue-js-going-to-take-over-react-in-2020-929c19806ac> [Accessed 22 December 2021].)* I feel like my decision will be okay as Vue is rising in popularity and already large companies have built their applications using Vue, showing its ability to scale up and work well for larger projects.

For the backend I am using a Mongo DB which I will connect to and query through Mongoose. Express will run the server. I quickly created a “server.js” file that would act as the entry point to my backend. I imported the relevant libraries (such as mongoose, express, body-parser and cors) which will help me connect to the server/database as well as help me handle the response data correctly and securely. I also made sure to keep plenty of comments in there explaining what the code is doing *(Figure 5).* I then quickly ran the script using “nodemon” and tested that it was connecting correctly to the server as you can see in the screenshots *(Figure 6)*.

Graphical user interface, text, application, email

Description automatically generated

Figure 7: Creating a cluster on MongoDB Atlas

Graphical user interface, text, application, email

Description automatically generated

Figure 8: Getting the connection URL for the server script

Graphical user interface, text, application, email

Description automatically generated

Figure 9: Whitelisting an IP Address to access the database

Next, I needed to connect to the database and therefore need to create it first. I chose Mongo DB and Atlas as it is free and on the cloud, making it ideal as it will require less upkeep from the developers (although that does come with the downfall of making it less flexible). To make the database I created an account and set up a Cluster in our region. Within that cluster I created the database “csc3131” and within that database initialised a collection called “users”. *(Figure 7)* Looking back, the name of the database was badly chosen as it doesn’t represent the data it is holding or the project at all and if it came to handing it off to other developers it wouldn’t make a lot of sense. Making the collection manually was not necessary as Mongoose would have created it for me when I first called the API and passed the database schema through, however I preferred to create it so that I had a visualisation of the set up for later. I made sure that the collection name was all lowercase and was plural as when I was researching Mongoose a common issue I saw was that Mongoose automatically made it lowercase and plural and it messed with a lot of people’s ability to retrieve data from a POST request because of a little miss-typing and I didn’t want to fall into that trap. *NodeJS, M., godage, P., Korkmaz, A. and Beliaev, E., 2021. Mongoose always returning an empty array NodeJS. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/14183611/mongoose-always-returning-an-empty-array-nodejs> [Accessed 22 December 2021].* Finally, I had to whitelist my IP address to have access to the database. *(Figure 9)*

A screenshot of a computer

Description automatically generated

Figure 10: Error occuring when connecting to the database

Graphical user interface, text, application

Description automatically generated

Figure 11: Screenshot of CurrPorts showing what each port is listening to

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12: The previous error is now resolved after commenting out a duplicate code statement

I tried connecting to the database however I got errors saying that I was already listening on that port. *(Figure 10)* My first instinct was to try and find out what was on what port as that would help me know what I need to close to allow the connection to the database. Something I was shown on placement was CurrPorts which works on Windows OS. *(Figure 11)* It shows what processes are running on each port, local address, remote address etc and it can be helpful to see if you have accidently created duplicates of your scripts or Docker containers. However, I couldn’t find anything listening on port 4000 (which I had changed it to so that it didn’t clash with the Vue instance on port 8080). This implied that it was an issue with the code, so I started removing lines and adding “console.log” statements to see at which point it breaks down. I have never used mongoose/express before so I am a little unfamiliar with the code to just be able to look and see where the error might be occurring. Finally, I noticed that I was calling the “listen” function twice and that was causing the issue. *(Figure 12)*

Text

Description automatically generated

Figure 13: Adding the database schema and creating the auth API's

Graphical user interface, text, application, email

Description automatically generated

Figure 14: Testing the auth API's using Postman

Graphical user interface, text, application, email

Description automatically generated

Figure 15: Testing the auth API's using Postman

Now that we were connected to the database, server, and front-end I could start on adding some API’s for the login feature and also start building the database schema for the user. *(Figure 13)* Since Mongo DB is non-relational, it is likely that I will only be using the one record per users and storing everything in arrays of objects instead of different collections (as sketched out in the design section). I feel like this will work to begin with as the amount of data will be small however I worry that this won’t be acceptable for large amounts of data, and it will become very slow – however for now I want to just be able to get some functionality started and then go from there and maybe reconsider when it comes to getting/storing the user’s items later. I made sure to implement error handlers into the API’s so that the responses can be displayed and debugged much easier. Since these API’s are to do with logging in/registering it means that the data needs to be handled securely – especially passwords – therefore I used JWT to help hash the passwords and generate a token that would act as the signal for a successful auth request. Finally, I tested the requests using Postman.

A screenshot of a computer

Description automatically generated

Figure 16: Added the navigation bar to the Vue template and built the router paths

A screenshot of a computer

Description automatically generated

Figure 17: Styling the navigation bar using CSS style tags

Graphical user interface, application, Word

Description automatically generated

Figure 18: Navigation bar layout complete

Afterward I had added a couple of test users to the database using postman I went on to start to build some components to the website. I started with the navigation bar and view router. Navigation bars have always been something I have struggled with and the same with the router in Vue. I ended up spending a lot of time googling and looking at examples/tutorials to better understand them and eventually produced the working one in *(Figure 18)*. I did this by using <router-link> components in view and the basic CSS for the styling and layout.

A screenshot of a computer

Description automatically generated

Figure 19: HTML for the Login form

Graphical user interface, application

Description automatically generated

Figure 20: Finished form template

I then went on to build the Login/Register forms and style them too. *(Figure 19) (Figure 20)*

A screenshot of a computer

Description automatically generated with medium confidence

Figure 21: Front-end methods to talk to the backend

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 22: SweetAlert (SWAL) popup to show a successful login

Now that I have the front-end forms, I can create the methods that allow the front end to talk to the back end. *(Figure 21)* I used a library called SweetAlert (SWAL) that generates simple and stylish pop-up boxes that are responsive to parameters passed through – so I made them interactive with the http response codes from the API call. For example, a 400 response will make the box say, ‘something went wrong’ and a 201 will mean that the login was successful, and the popup will say ‘success’. *(Figure 22)* The reason for using SWAL was because it meant there was uniform responses for the user, and it also had an animation that looked more professional than anything I know how to do at this point. I also used it because I am beginning to fall behind schedule having spent too long trying to understand the Vue router so there is a chance that some features will not be completed on time.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 23: Global variables to use in CSS

Graphical user interface, text, application, email

Description automatically generated

Figure 24: New navigation bar style and layout to match the design sketches

Graphical user interface, website

Description automatically generated

Figure 25: New form styles to match the navigation bar and design sketches

A picture containing calendar

Description automatically generated

Figure 26: Board Page design completed ready for data to be injected into it

Despite previously saying that I was running behind on time I got feedback from peers on the layout of the navigation bar and knew that I needed to work on it a bit more. It was never finished and the style in *(Figure 18)* was far from the designs and storyboard drawn up but I was happy that it had the functionality for the time being. It needed to be sharper and take on a more modern design/colour so had put aside more time for this.

I created some SCSS Mixins and moved a lot of the styles to the app component (global) so that they can be applied across all components and make the whole front-end much more modular and smoother. I also worked on the navigation bar to get something much closer to the designs. However, this took up a lot of time and may affect the completion of other features.

Graphical user interface, text, application

Description automatically generated

Figure 27: New schema and API call

I updated the schema for the database to have two arrays of objects. One to hold the details of each board the user has and then another to hold the contents of each board. Since Mongo DB is non-relational and is for unstructured data I am using the arrays to act as collections in themselves with their own IDs to link them together. Since there is no SQL query’s for a Mongo DB this should be an adequate alternative however these arrays could get very large and it would not be good if the application needed to be scaled up. Once the schema was updated I created another API to get the details from the first array and display them as seen in *(Figure 27).*

## A screenshot of a computer Description automatically generated

Figure 28: Created a AWS S3 Bucket

It is unrealistic to store images in the mongo DB and therefore I am using a AWS S3 Bucket. These allow for unstructured storage for a very large amount of data so it is the ideal way to store the images a user may upload – the mongo DB can then just store the source path to the S3 bucket. Therefore, I created an account and set up the bucket and managed the permissions to make the objects accessible.

# DevOps Features

### Version Control & CI/CD

Graphical user interface, text, application, email

Description automatically generated

Figure 29: Add branch rules to the github repository

Graphical user interface, text, application

Description automatically generated

Figure 30: Created CI/CD features in Github Actions

Throughout the development process I have been committing my code to a Git Hub repository periodically. To begin with I was using the main branch but once production got underway, I was able to switch to a better design. Instead, I would create a ‘feature’ branch and work on that until said feature was complete. A pull request for this would then be made and in an ideal world, other members of the team would read, review, and approve the pull request (I set up these requirements in the GitHub settings). Once it was approved, it would then be merged into the main branch. In addition, I also created a CI/CD Workflow in GitHub Actions that builds the application, runs lint to check for uniformed code, and also runs the test files. This is called on every pull request and it has to pass before the feature branch can merge. This will help people working on the project to ensure that their code isn’t going to break the entire thing. I didn’t get a chance to make the workflow as complicated as it should have been and this is something I would expand on further if I had more time – for example, I would run tests for different environments, run smoke tests, code coverage tests, etc.

### Maintainability

A screenshot of a computer

Description automatically generated

Figure 31: Comments in the code

I have included comments throughout my code to help other developers. In addition, I have also tried my best to keep my code tidy and modular. *(Figure 18)* Vue.js helps with this as the framework requires you to work in small components but within those components I have tried to use the correct syntaxes – this is backed up by the lint test in GitHub actions as this would fail if the code wasn’t properly following the rules. For styles I decided to put them all in the <style> tags and not to have half inline half external as this will make it easier to maintain and update in the future.

### Security

I haven’t done as much in this area as I should have. I have hashed the passwords in the database and also made it so that you have to have your IP whitelisted to be able to access the database which for the time being stop unauthorised users getting access to that data however it complicates it for deployment and that will have to change.

# Reflection & Evaluation

I reflected on some of my choices in the implementation section as I went along but overall, I thought that this project went mediocrely at best which is hard to admit. I was unable to finish a lot of the features of the application and the DevOp features I have implemented are small. If I had more time to learn and better understand how to implement these features I want to say that I could finish this as I was starting to progress much faster towards the end. I was slow to progress at the beginning and focused too much on the wrong thing and lost a lot of time and motivation due to it. In the future I will better plan for this by setting deadlines for myself for a feature and creating a schedule that I could keep to. On the positive side I have learnt a lot about the new languages and tools I have used throughout this project and am now much more aware of the work that goes into maintaining a project.

# Other

MongoDB: U**sername:** admin **Password:** adminpassword