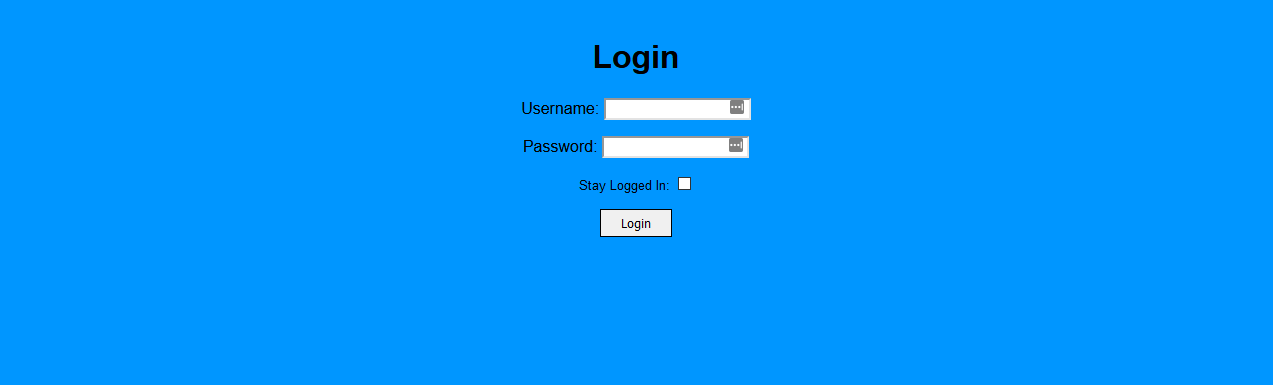
Report – Finance Application

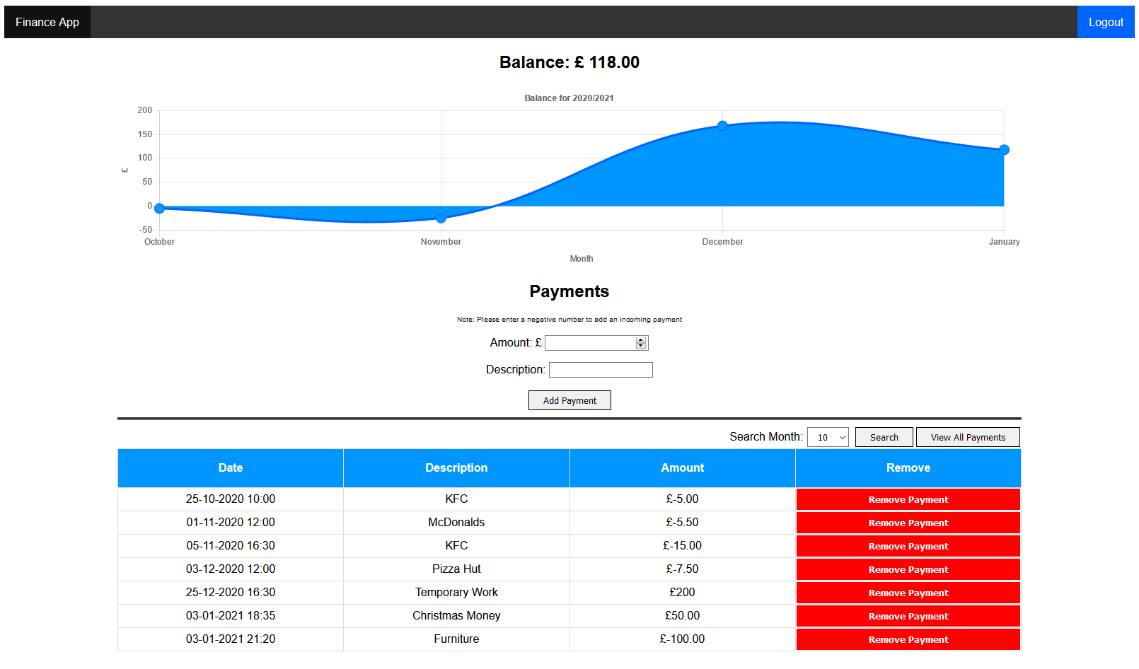
Video: <https://youtu.be/M1iJXDnJvaA>

# Requirements

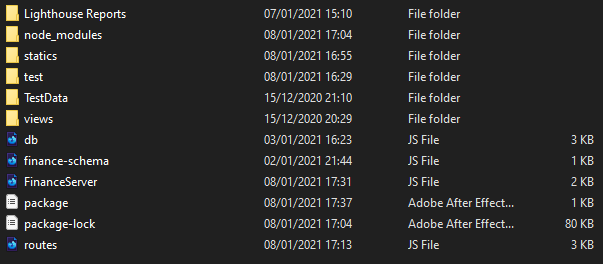
My finance application is aimed at both business and individuals who want an easy way to be able to manage their finances. It includes 2 pages, a login page where a user logs into an account and a home page where all the finance information is shown. The application would be useful for many users, as it creates an easy way to be able to track the balance of the account over time, which for business and individuals would be helpful in seeing how much money they are or aren’t able to spend at the current time. All the functions of the application are laid out individually and in a user-friendly manner, which makes it extremely accessible, needing very little training, if any to get a user up to speed with using the application.

The main features of the application are being able to see balances for each month as well as the current balance, being able to see all previously made transactions whether they be into the account or out, being able to remove payments from the system and being able to add new payments. All of these features are essential to making the application function properly, making it easy to manage the user’s finances whether that be for a business or just an individual. The application also features a login system, which ensures that these finances are kept secure. This is essential for any application dealing with finances as huge amounts of money would be going in and out of the accounts and any malicious activity would be extremely bad. Another feature that applies more to businesses is the use of WebSockets, here WebSockets are used to alert other members currently using the application that a payment has been added or removed, which also adds it on their page. This is very helpful for a business, where multiple people could be using the application at the same time. This feature helps to mitigate the chance of multiple people entering the same data into the application at the same time and creating duplicate payments.

One other very important feature that I implemented is a local file to store whether the user wants to be kept logged in, as well as if they are logged in or not. This stops users from navigating straight to the home page without logging in, which would be a massive security problem. The keep logged in function on the other hand is just an improvement of life change. This keeps the user logged in at all times, this solves the issue where they navigate to the login page without logging out of the account. Normally they would have to login again, but with this ticked when logging in, it automatically redirects them to the correct page instead.



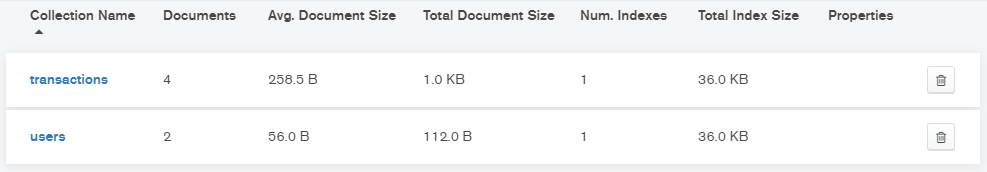
# Design

I made sure to split all the relevant functions out into their own files and folders. Below you can see all of my main server files are in the root of the application, the views contain all the page templates, and the statics contain the CSS and images for the application. The structure of these make it easier to find the relevant files quickly, it also helps to discern the files apart, as only one type of file will be in a folder. The file structure would also be important if another user were to add something to the application, as everything is clearly labelled and laid out.

Communicating with the database is only dealt with through the db file which also uses the finance-schema file to regulate the data names and types. The FinanceServer file is the main node server, containing all of the GET and POST requests for the URL, all of these then go through the routes file which decides what to do. All of these conform to the regular RESTful standards with all of the GET requests loading a page and resources on it and the POST requests manipulating the data in some way whether this be updating or creating documents.

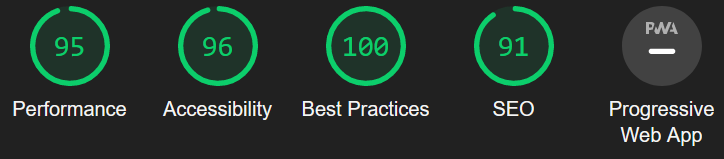
In terms of code structure, I managed to keep it consistent throughout the application. This includes consistent camel case used for variables and how the code it is set out within the files. For example,

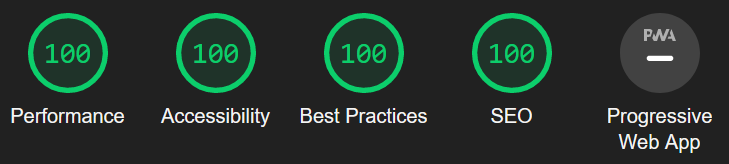
The database that I use for the application is split into two collections, one for the transactions and one for the users. For the transaction’s collections, each transaction is a month, year and balance. Inside of that I then have an array which stores the payments for that month & year. As I only had to accommodate one user, this was quite easy, however if I made the application again, I would have found a better way of storing the transactions as currently every user shares the same ones. For the users table, this stores the username and the password for the login screen, while this is stored in plain text, which doesn’t make it very secure, the data is never passed to the user as it is just filtered so this is only a problem for employees using the database. My database and structure can be seen below.

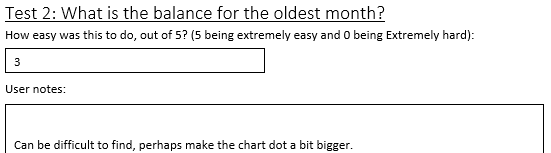


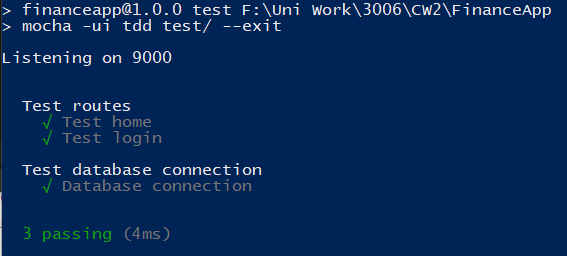
# Testing

I used Google Lighthouse testing to ensure that my application didn’t have any underlying issues, was performing well and to check for usability issues. After running it the first time, there were a few small issues that came up so went through each one, changing them and then reran the tests. This included items like the page language and the size of some of the buttons to optimise it for mobiles. I ran Lighthouse on both the Login and the Home pages of the application to cover the whole app and got to the below scores (1. Home, 2. Login). For the home page, the only items I would change going forwards is to locally store the both the Jquery.js and chart.js files, as they are the only items limiting performance.

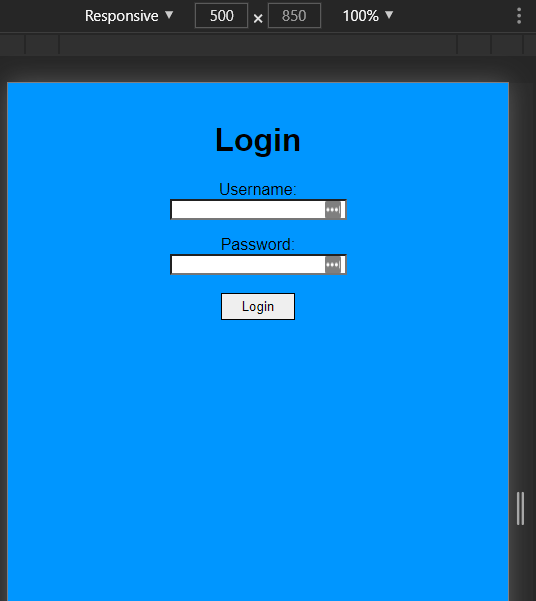
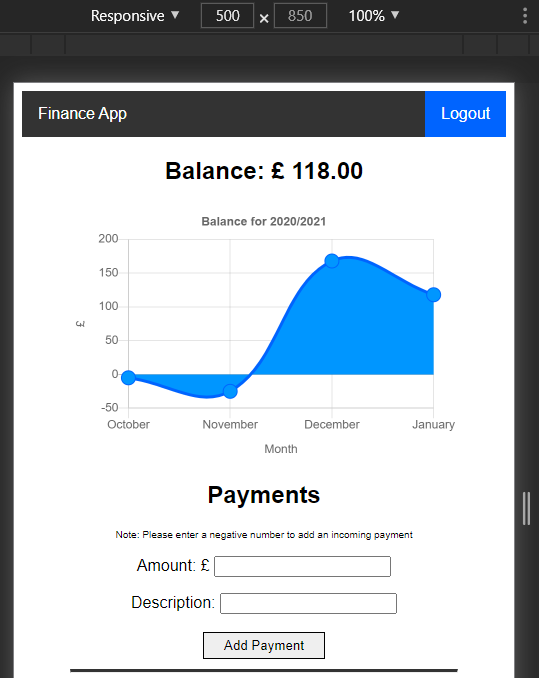




I managed to get a couple of users to do some usability testing on the application. This included testing the main functionalities of the application, as well as reviewing how easy it is to find specific items on the page. This was extremely helpful to me and I made changes to the application following the user tests. Some of these changes include changing the months from digits to text, making the chart dots bigger as well as changing some colours around to make items stand out as well as some other minor changes.

I also created some minor tests in Mocha to check the routes for the main pages as well as the connection to the mongo database. These tests were run before every commit, to check the main functionalities hadn’t been altered between commits. This was very important as without these running properly the whole application would be unusable so user testing wouldn’t be able to be completed fully.

I also tested using Chrome Dev Tools, making sure that the application works and is fully usable on mobile devices. Below you can see me testing the pages down to 500px wide and it still being completely usable.



The way that I tested worked very well as I didn’t just test at the end, I continuously tested throughout the development of the application. This meant I wouldn’t get to the end and have to make a massive change to a fundamental part of the application. It also meant problems could be fixed as soon as they started to occur, so that I could spread bug fixes out rather than fixing all the problems simultaneously.

# DevOps Pipeline

To develop my application, I used both GitHub and Heroku. This meant that I could progressively add to the application, developing and then testing new features, without it actually going live. After I had tested a new feature that I had added and made sure it worked completely, I pushed it onto Heroku where the application could be used by a user. This worked well as I didn’t have to worry about somebody using a broken site, having committed a change the application and then there being a massive bug that breaks it. This was extremely important as I did remote user testing on my application through Heroku, so if I had deployed to Heroku after every commit, a user might have reported a problem that had already been fixed.

For my development environment, I ended up working on the project locally on my device and then only committing to my GitHub after I had checked everything was working correctly and the change, I had made didn’t alter the application in an adverse way. If I had done this project again, I would have created another GitHub branch to hold the changes, as this would have been a lot safer and made the development cycle run smoother in the long term however the way that I did it still worked well.

# Personal Reflection

If I were to make the application again there are a few things that I would change, this includes linking the finances (balance, payments etc) to each individual user/company as currently all users show the same data and changing the popups for the payment added/removed as currently when another user adds a payment there is a generic popup for them which doesn’t look great. If I were to do the sockets again, I would have them as a small banner or popup not in the middle of the page so the user could carry on with what they were doing without getting disturbed by the popup.

Another small change I would make is changing the way that incoming payments are added. Currently the user has to enter a negative number for it record an incoming payment, but this isn’t very user friendly, so perhaps having a section for incoming/outgoing payments or a tick box would make it easier to use. Alongside this, a date picker for adding payments could also be a useful feature as it would allow the user to enter them for any period of time rather than the data automatically being generated from when the payment is added.

All of the technologies used worked well and enhance the user experience, however I only used the minimum needed for the feature in hand, as adding too many dependencies slows the website down considerably. The chart for example gives the user a quick and easy way to be able to see their spending habits, without it, the user experience would be drastically worse and the trade-off on speed wouldn’t be that noticeable.