SWD503 Enterprise Database Development - Report

During this project, I have designed, developed and deployed the Foster Carers Portal.

The code for this project is available on my Github repository.

A demo user with some data pre-populated has been set up with the following credentials.

> **Email:** 4cartd62@solent.ac.uk
> **Password:** FosterCarersPortalDemoUser

Introduction

This project has addressed the following problem.

> As a foster carer, we are we required to supply information to the child's social worker on their what they have done and how they are developing. Currently, this is done through the use of paper documents, word documents, spreadsheets and emails, which is inefficient and prone to the loss of data. The ideal solution will allow all data to be logged securely against each child by a user and viewable and recalled easily.

The primary issue that this project seeks to resolve is to consolidate the storage of all information related to the children in our care. in terms of data we approximately log the following per month:

- 200 feeds
- 200 changes
- 20 appointments (+ expenses)
- Adhoc incidents/accidents
- Weekly Reports

System Overview

Users of the system are required to be logged in to view their sensitive information. No data is available without logging in.

Once logged in a user can view all the children in their care and can add, update or remove child records and activities related to each child.

The salus database contains eight collections for the user, their children, and the children's activities. This data structure has been devised so that all child activities belong to the child (the childID is included on all activity documents). To ensure that only the user can access these records, all child and child activities record the userID in the owner field.

The Child activities (Appointments, Feeds, Changes, Incidents and Reports) each have a single page which gets a list of activities when the page loads. Creates, updates, reads and deletes are performed using the API controllers.

Expenses are the only area which doesn't link directly from a child as these are linked to appointments, but work similarly to the other child activities.

Lastly, the 'My Expenses' page shows the user all of their expenses across all children, with the option to download the data, so that the expenses can be submitted.

Key Design Decisions

I have chosen to use Nodejs/Mongoose/MongoDB for this project as there isn't an existing structure within the data to work and this made making adaptations during development easy.

When developing this site, I have considered the ability to have multiple users log in and access their records, as such in all collections I have stored the owning userID to query against.

The primary concern with this project is the sensitive data which the system holds, and that access should be restricted by User. As such I have used cookies to record the user ID after they have logged in. This allows me to check that the user is the owner of the child record that they are querying against. By doing this I can prevent data being read or amended by other users, who do not owner the relevant documents.

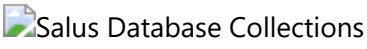
I have attempted to reduce the number of page loads a user would experience, but having the Child Activity pages use API to handle Create, Read, Update and Delete actions and add the resulting data to the page through javascript.

One of the most time-consuming aspects of the problem is the collation of expenses as it is required to collect details about appointments to be able to submit expenses. Therefore, I have implemented a download function on the "My Expenses" page to allow for a user to download their expenses across all their child's appointments.

Another required piece of functionality is a search function on reports to be able to check back against reports previously written reports. As this helps identify patterns in behaviour to communicate to the social workers.

Database Design

The below image indicates the data which is stored within each collection in the database. I have indicated in brackets where the Id of other documents are stored. To indicate how they are linked.


Salus Database Collections

All documents will hold a reference to the user, such that data can be filtered appropriately by User. I have also included the Child ID on all documents which relate to a Child such that these can be queried effectively.

When it came to Expenses, I had initially planned to store this data as a subcollection on the Appointment document. However, to be able to easily download all expenses, I instead opted to set up expenses as a separate collection with reference to the parent Appointment. But to be able to show how many expenses had been created for an appointment I chose to also store the references to the expenses in the Appointment's Document. As a result, this means that when an Expense is created or deleted, the Expense Id is also added or removed respectively from the parent Appointment document.

Security

Due to fact that the data stored is highly confidential, security has been a leading consideration when developing the solution to this problem. As such when accessing the site a user needs login to view any

information. Passwords in the database are hashed using BCrypt and hashes are compared to establish if the user can log in, once logged in a cookie is created to record the User's ID and the cookie is set to expire in 14 days from the last interaction.

Before accessing any data each request checks that the user is logged in, and then checks that the user has is the owner of the document which is being accessed.

Scalability

Whilst I have been addressing a personal problem through this project, I have continued to be open to other foster carers using this platform, so I have ensured that more users can be added at any point without impacting the security of the system or data.

However, I am aware that a limitation with the system is that all related data is displayed when looking at child activities (particularly feeds and changes) once these run into hundreds or thousands of documents, then viewing these will be cumbersome for the user. Therefore in the future, I will be looking into adding facilities to filter the data and limit the data rendered through pagination.

Conclusion and Reflection

Overall I have enjoyed developing the solution to this problem, and it is effective at allowing a user to record all their records in one place. The auto-population of dates and times also encourage the users to enter data, close to the time that the activity happens.

Throughout this project, I found that the initial setup of NodeJs and the additional dependencies of the project did take a while to first get set up, but once the setup, the development was quick and it was easy to make amendments as the development continued.

I have not previously had much experience with handling dates in javascript and found this an initial hurdle in development until I had got the how the date should be formatted for viewing, submitting and populating input fields. Additionally, as Nodejs only includes the American DateTimeFormat by default I had to additionally download and run the full-icu for the Intl module, this then allowed for English formats to be used.

I have found that working with data in a non-relational way was a challenge and found that my thoughts often lead to thinking about how data could be optimally stored rather than quickly retrieved. However, the turning point in this project for seeing the advantage of MongoDB was when storing data against both appointments and expenses and the simplicity that this approach allowed. I do however think that I need to further experiment in the future with how to best record relationships between documents as these can be stored on either the parent document, the child document or on a separate relationship document, or a combination of all three.

Reflecting further on the relationship between Expense and Appointment, I think that rather than holding an array of expenses, I should have limited appointments to only have one expense per expense type to limit users from creating duplicate expenses, which the current configuration would not automatically highlight.

Another area which I feel I could improve upon is regarding error messages back to the user. These are currently quite lacking in the system so error checking can be frustrating to users.

In the future, there are many aspects I would seek to improve this project. Firstly I would like to include attachments such that these could be linked to incidents (e.g. photos) or expenses (e.g. receipts), this would go further in the ensuring that all aspects can be done in one place. This project could also be improved by implementing a sharing facility such that users could share access to child records (e.g. the child's social worker). This would then mean that data would not need to be sent via email.

Once records were able to be shared, then an email notification system to automatically alert the social worker when an incident or report has been created could also be beneficial.

When I have been developing this I have had in my mind that all activities will be unique to a child (this is true in my case as we only foster one child), but other foster carers may foster siblings, so appointments like contact would be shared, so the appointment should relate to both children.

Concerning MongoDB and Mongoose, I feel that I need to continue to learn and use these, as I would have liked to make more use of the pre and post actions to update related records rather than manually writing these from the controllers. The advantage of this is it would improve the simplicity and reliability of the code.

Lastly, I would like to improve the user account management of the system, as this is currently very lacking. I would also look into the use of oAuth to simplify password management for users.