Medical Sonography ICT Capstone Project

Duy Quoc Anh Nguyen

Rijin Reju

Contents

Codebase overview:

1. Directory structure:

- /my-app : contain all frontend source code
- /server: contain all backend and sever source code

2. Important directories and files:

- /my-app/src/routes: contain all main webpages
- /my-app/src/index.js: is the entry point for our application, where we define routes for the whole website.
- /server/app.js: is where we set up database and database request.
- /server/sql.sql: is the file we run any CRUD queries for database.

Environment setup:

1. Node.js v18.x
2. SQLite3 v5.x

Code walkthrough:

Logbook page: logbook.js

This page is designed for students to log scans. There are two essential parts: log the scans and present the scans.

- Logging scans: When a student fills in scan's detail, each data input will be stored temporarily. After student finish filling in and submitting, a form will be created to store that information in the database. All these will be stored upon original fetched logbook using array (all original scans in database + temporary scans that has just been submitted).

- Present scans: using the second array (modified from the first logbook array to sort all scans by date and store all information in this array) to present those scan's detail in the table.

This page also linked to two other pages which is : PlanForImprovement.js(where student fill in their plan after supervisor submitted their comments and performance appraisal) and Milestone.js (when student reach a milestone, they will need to fill in the milestone form).

Student Milestone document (Performance Self-appraisal): Milestone.js

This page is a digital form for milestone document. All student's and list of supervisor's detail are fetched and prefilled in the input field (some are disable). Student then choose who (supervisor from the list) will receive the milestone document.

Data from JSON file is also loaded into a table to present all questions and answers. Answers later will be stored as a text along with other information such as student's signature in database after student submit the document.

Email notification is implemented to send to a specific supervisor that student choose to be in their milestone document.

Supervisor milestone document (Performance Appraisal): SupervisorMilestone.js

Supervisor will be able to access the in-process milestone document through Supervisor home page (Supervisor_Home.js) and a list of their student (AcceptedStudent.js). Choosing a specific student's milestone document will lead to another page(MilestoneAcceptedStudent.js) where it shows a list of completed milestones document and a document that's in-process (Performance Appraisal document).

When accessing in-process document, the page will show all student's detail (student that chose supervisor), student's answer (Performance Self-appraisal), a new milestone form for supervisor, and an additional comment text area. After submitting supervisor's detail (including signature), and all documentations will be store in the database.

Student's answers (milestone document answers) are loaded and modified from original JSON data (fill in the original data with the answer accordingly)

Student's Plan for improvement: PlanForImprovement.js

When supervisor finish their document and comment, student will be notified to go to this page. In this page, it will be loaded and presented with all supervisor's and student's information from previous document (answers; each individual details and signature; date

supervisor gave consent etc..). A new text area for student's plan is provided and date of consent is required to input. After submitting, this information will be store for milestone document review later on.

Supervisor's and student's answers are loaded and modified from original JSON data (fill in the original data with the answer accordingly)

Full Milestone document: MilestoneSummary.js

All information that related to milestone document will be shown in this page: two tables for questions and answer from supervisor and student; supervisor's comment; student's plan; consent date from both supervisor and student. They are fetched by linking multiple tables in the database (milestone, milestonedoc, student and supervisor table).

All answers are loaded and modified from original JSON data (fill in the original data with the answer accordingly)

Course page: CoursePage.js

This page presents all functionalities that are related to logbook and milestones, with the side with some basic announce from the course/program. By accessing each function/button it will link to different pages: Manage Logbook (Logbook.js) and View Milestone Documents (MilestoneSummary.js)

Registering account page: Register.js

In this page, there will be a card and inside the card there are multiple input field that used to register account. Depending on which role (student or supervisor), input fields requirement will be changed based on it. All input fields are required to fill before submitting. After submitting, information will be stored in two tables: account and student/supervisor.

With existing email from supervisor (student have request supervisor to be in their list of supervisors using supervisor's email, that email will be temporarily stored in the supervisor table), supervisor will be able to update with new all information using that email and new account will be attached to supervisor.

Student home page:-

Student home page is fully covered through Student_home.js . This is the page students would see once they login. Everything within student_home.js ensures what happens within their respective page student pages.

```
      <button style={buttonStyle}>General Logbook</button>
      <Link to={`/CoursePage/${studentID}`} style={{ textDecoration: "none" }}>
        <button style={buttonStyle}>Cardiac Logbook</button>
      </Link>
      <button style={buttonStyle}>Vascular Logbook</button>
      <Link to={`/Supervisor_Details/${studentID}`}>
        <button style={{ position: "absolute", top: "40px", right: "10px" }}>
          Supervisor Details
        </button>
      </Link>
      <Link to={`/Current_Supervisor/${studentID}`}>
        <button style={{ position: "absolute", top: "40px", right: "175px" }}>
          Current Supervisor
        </button>
      </Link>
    </div>
```

The buttons shown above are implemented within the student_home.js , these buttons would also containt the parameter of the student id, as each student will have their unique student id number.
The buttons have respective links which will lead them to different pages such as Cardiac logbook, supervisor details and current supervisor.

Please note :- Vascular and General logbook button does not work.

Sign in page for students:-

Signin.js :-

This is the specific page used for students to sign with their student credentials that is there within the database, Within the student sign in page, we do ensure if the student credentials match within the database.

```
try {
      const response = await axios.post('http://localhost:8081/validateStudent',
{ username, password });

      if (response.data.validation) {
        setLoginSuccess(true);
        setError('');
        setStudentID(response.data.studentID); // Set studentID in state
        console.log('Student ID:', response.data.studentID);
        console.log('USERNAME:', username);
        console.log('Login successful');
      } else {
        setError('Invalid username or password');
      }
    } catch (error) {
      console.error('Error logging in:', error);
      setError('An error occurred while logging in');
    }
```

This check within the database to ensure the student username and password matches , if it does not match it will get out a invalid username or password error
this is the post request which signin.js would compare with.
The user then submits with their username and password within the login page. If the login is successful it will store the student id of that student in the react component state for the future use which would be beneficial as each student would their own supervisors and logbooks which would be unique to those students.

```
app.post('/validateStudent',(req,res) => {
    const {username,password} = req.body

    db.all(`SELECT * FROM ACCOUNT JOIN STUDENT ON ACCOUNT.accountID =
STUDENT.accountID
```

```
        WHERE username = ? AND password = ? AND role = 'STUDENT'`, [username,
password], (err, rows) => {
         if (err){
            console.error(err.message);
            return res.status(500).send({error: 'An error occured'});
         }

         if(rows.length > 0){
            res.json({ validation: true, studentID: rows[0].studentID });
         }else{
            res.send({validation: false});
         }
    });
});
```

For Staff Sign in page:-

username, password, error and login are state variables used by useState which will be visible from the SigninAsStaff.js page.
This page would be dedicated for the use of supervisors, staff members(UniSA staff members) and admins of the page.
From a login perpepective , login would send a post request to the \validateStaff endpoint if the username and password is correct. (Username and password needs to exist within the database) if the credentials are inncorect it will give the user an error stating access denied.

From a backend perspective of \ValidateStaff

This is a post endpoint which will compare the username and password within the database and alongside compare the roles, the roles of the user has to be either a staff, supervisor or an admin. If the role does not match with the above 3 then the error would be access denied.
So this would ensure only authorized users can access the page even though the page exacts same as student sign in page.

The handleSubmit function within signinasstaff.js will be the component that sends a post request to \validatastaff with the inserted credentials by the user.

```
app.post('/validateStaff', (req, res) => {
    const { username, password } = req.body;

    db.get(`SELECT * FROM ACCOUNT WHERE username = ? AND password = ? AND (role =
'STAFF' OR role = 'ADMIN' OR role = 'SUPERVISOR')`, [username, password], (err,
row) => {
        if (err) {
            console.error(err.message);
            return res.status(500).send({ error: 'An error occurred while
checking the credentials' });
        }
        if (row) {
            return res.send({ validation: true, role: row.role, accountId:
row.accountID });
        } else {
            return res.send({ validation: false });
        }
    });
});
```

```
      if (response.data.role === 'SUPERVISOR') {
            navigate(`/Supervisor_Home/${response.data.accountId}`); //
Navigate to SupervisorHome if role is SUPERVISOR
          } else {
            navigate('/Staff_Home'); // Navigate to Staff_Home for other roles
          }
```

The above would ensure that staff members would be led to the staff page based on their role and supervisors would be led to the supervisor bage which will be compared within the database. Each role would have their own significant pages to manage their students.

Staff Home Page:-

Staff_home.js is the page used for staff home page perspective, everything within this page controls the staff home page.
Staff home will showcase all the students within the page. It extracts all the students from the student database particullary and displays in a tabular format.

The database itself contains a coloumn for students and this students database will have detail such as studentid, name, email,account id, phonenumber and asar.
Everything in the table for student database would be shown in the staff home page except for the student's accountid number as that is not needed for staff member to see.

```
app.get('/getAllStudents', (req, res) => {
    const sql = "SELECT * FROM STUDENT";

    db.all(sql, [], (err, rows) => {
        if (err) {
            console.error('Error fetching all students:', err.message);
            return res.status(500).json({ error: 'Error fetching all students: '
+ err.message });
        }
        res.status(200).json(rows);
    });
});
```

This request shown above does the job of fetching all students from the student table. /getAllStudents is then used to fetch the request from the staff_home.js to fetch all the students from the table.

Supervisor Home page:-

Once the supervisor signs in, their homepage would consist of 2 options, one for viewing their pending student requests and the other for viewing their accepted students.
Pending student requests would showcase the students who uploaded their details in the supervisor_Details.js.
If the email matches with the credentials of the supervisor it would be then linked to the databased using the supervision table . Supervision table consist of student id, supervisor

id, and issupervisor coloumn(boolean) From a supervisor home page perspective they can view the pendingstudentrequest which will be from supervison table which matches alongisde their supervisorid. The supervisor then has the capability to either accept or reject the students.
If accepted , the issupervised coloums turns to true (boolean (1)) within the supervison table. And if rejected its then removed.
Once the supervisor accepts the students the students would be visible within the accepted students page.(AcceptedStudents.js) and all the pending student requests can be visible from the PendingRequests page (PendingRequests.js)
There are 2 buttons which would lead to either or pages as shown below.

```
      <div className="container text-center">
            <h2 style={{ padding: "30px" }}>Supervisor Dashboard</h2>
            <div className="d-grid gap-2 col-8 mx-auto">
                <Link to={`/PendingRequests/${accountId}`} className="btn btn-
outline-primary">
                    Show Pending Requests
                </Link>
                <Link to={`/AcceptedStudents/${accountId}`} className="btn btn-
outline-primary">
                    Show Accepted Students
                </Link>
            </div>
```

Supervisors homepage also uses a get endpoint request /getsupervisedstudents/:accountid , to display the student id that matches with the supervisor's id. Accept or reject button would manage the handleDecision function. A felexibility is given with the supervisor homepage where in the future you could potentially implement the pendingstudentrequest and acceptedstudents just from the same page instead of jumping into different pages.

PendingStudentRequest page:-

The pending request page will be the page where supervisors can either accept or reject the students which are linked through the supervision table, upon accepting the

issupervised coloumn within the supervision tables turns to true, ensuring the students are then visible and linked in the database.

acceptstudent or rejectstudents determines the endpoint decision which would then send a post request to the backend if its successful. The backend then updates the supervison status. Once the students are accepted they are then removed from the pending student list and would be then visible at the accepted student page.

AcceptedStudents.js

The acceptedstudent page would be the page that would display all the students that are accepted by the supervisors decison from the pending request page , this fetches on to the database within the supervision table which contains the coloumn of issupervised , if the issupervised coloum is accepted(true(1) Boolean) then the students would be visible on this page.

For each accepted student their details such as student id, name , email and the milestone document would be then visible to the supervisor. Supervisor would be able to click onto the milestone document which would lead to the milestone document page where the supervisorID and studentID would be used as route parameters.

CurrentSuppervisor page

Current suppervisor is a button that visible on the student homepage, where students can view their current supervisor.
Current_Supervisor.js is responsible for this page, the page itself displays the supervisors details such as supervisor name, email and qualifications of that speicfic supervisor.
This page once again fetches onto what is given in the supervison table but would not display the information unless the supervisors has accepted the student , Once the supervisor has accepted the student the issupervised coloumn within the supervison table turns true which would let the students view their current supervisor who has accepted them and the same logic is applied for the acceptedstudent page from a supervisor perspective.
Student's cant edit the details on the current supervisor page as the page's main intention is to only let the students see their supervisor (Read only mode in general)

Supervisor detail page

Supervisor detail is another button the student's have on their student homepage, the intention of this page is for students to upload their supervisor details, students are allowed to upload/confirm their own supervisors as part of their course. This is done through the supervisor detail page. The page itself lets students insert their supervisor'semail, name and qualification and there is also a note given under the confirm button to let the user's be aware that there will be an email sent out to the email address mentioned in the email column of the page once they hit confirm.
Students have the flexibility to upload as many supervisors as they want and there is a button under the "Note" called "Add more supervisors"
Clicking on that button would allow the students to fill up the same box (name, email and qualification) allowing them to add more supervisors.
Once the confirm button is clicked an email would be sent out using the email js, it is a front end server that we have made use off for this project. The front end server comes with service id, template id. The email can be sent to any email service you have selected , we have selected gmail for our testing purposes and we had also made a temp account called sonographymedical@gmail.com just for sending out emails. As all emails sent out would be linked to one specific gmal account(sender) , there's no restriction to reciever as long as its a gmail account and the gmail account actually exists.
Emailjs comes with a quota usuage of 200 emails per account , once the quota is completed you would then have to pay money to continue with the service or make an alternative account.
Given below is the section that would deal with the email js server within Supervisor_Details.js page.

```
// Send email notification
            emailjs.send('service_otfz7pr', 'template_g28nptg', {
```

```
                    name: supervisorName,
                    to: supervisorEmail,
                    from: 'sonographymedical@gmail.com',
                    body: `Hello ${supervisorName},\n\nPlease confirm your
details on the UniSA Medical Sonography website.`
                }, '9So_wYKnefvG9Mdqr')
                .then(() => {
                    console.log('Email Sent Successfully');
                    setShowAlert(true);
                })
                .catch((error) => {
                    console.error('Error sending email', error);
                });
```

Once the confirm button is clicked the email is sent out and also the details of the supervisor is saved temporarily into the supervisor's table and it will be saved in the supervison table alongside the student's student id. So, supervisors table would consist of the supervisor information the student inserted , supervison table would do the job of matching the id of both student and supervisor but with a default value of false in the issupervised coloumn of the supervison table.
Only once the supervisor logins and accepts the student will the issupervision coloumn turn to true.
The email is set to be unique so students cannot edit any details of the existing supervisors within the database.
If the supervisors already exist and have an account in the database and if a new students put those supervisors details in the supervisor detail page that would not edit any information of the supervisor as long as the email matches in the database.
The supervisor can then login and still accept or reject the student regardless of any spelling error the user makes while uploading the supervisor detail.
Email is the main part of this page as that would be checked in the database if it does exist.