COS70008 Technology Innovation Project

Semester 2, 2022

Project Report

Lucas Qin

Individual Assignment

Course Code: 70008

Tutor: Dr.Naveed Ali

Student ID: 103527269

Student Name: Lucas Qin

Word Count: 5306

Date: 24/10/2022

Executive Summary

This article introduced the Corelia project first, then reviewed the conceptual design I have done using an evidence-based approach. Meeting with clients helped us understand that we should focus on two innovative ideas, search function and community forum page. Then I reviewed my contribution as well as learning outcomes. I introduced the pages and functions contributed by me in detail. I also write some reflections on what I have done well and what I could do better next time. At the end of the article, some future recommendations and extra information like technical logs and links to Github are provided.

Table of Content

1. Introduction

1. Project background

It has been a long time since people forgot the female composer's contribution to classical music, while many of the pieces created by female composers are out there. The Corelia project aims to collect those pieces with information like composer information, recording of the piece, musical score (if applicable), etc. Those data collected will be used to promote women composers and their works, and encourage musicians and concerts to diversify their choice of repertoire. To achieve this purpose, the Corelia project wished to build a website that served as a centralised database for classical music players to discover women composers' works. Apart from the database, the Corelia project also wishes to provide a forum for its users to communicate with each other, including commenting, posting, sharing and collecting a post.

2. Project motivation

This project has many aspects that interests and motivates me. First of all, the Corelia project is a real project with real clients, the project in its development lifecycle will receive real feedback from clients, that could help me to learn how to communicate with clients and understand client's business, and what client's needs. Secondly, the technologies that the project will be using are very up-to-date, that means I will work on some technologies that I can use on other projects. For example, NoSQL has many advantages when compared to relational databases, including high flexibility, high performance, high scalability, etc. Hence NoSQL can deal with large amounts of data with high speed, it will become more and more popular in the future. Thirdly, the project is a build-from-scratch project, that means I can start planning, designing, developing, testing and deploying the solution from nothing. That would help me build ability and confidence in cooperation with other team members, learn and apply new knowledge to a project.

2. Design Concept

1. Problem statement

The Corelia project is seeking to build a website with the following features. Firstly, the website will include all information on composers and their works. Secondly, the website needs a blog feature that provides recommendations to composers and their works. Thirdly, the website needs a forum function for users to interact with each other, or share, comment on others posts. Lastly, the website needs a powerful search function that allows users to search the website contents.

2. Methods/Methodology

Literature review and Sprint Design Methodology has been applied to the conceptual design. There are five steps of the Sprint Design, Empathise, Define, Ideate, Prototype and Test. More details can be referred to assignment Conceptual Design.

3. Design constraints

The design comes with several constraints. First of all, the design was based on limited information provided by clients, the project description on Canvas missed a lot of details and there was only a very short meeting with clients to discuss the first version of design. Hence there was no time to discuss the client's rationale about certain requests and other options. Secondly, the design was limited by school schedule hence there was not much time for the design. Thirdly, the design was restrained by the team's design ability and professional skills. Fourthly, the design missed non-functional requirements on the design. Fifthly, there were no users recruited for usability tests due to lack of relevant resources.

4. Specifications

The design has the following features:

1. A homepage with recommended composers, repertoires and instruments on sliders.
2. A composer page that displays all composers' profiles, including their works.

3. A repertoire page that displays all composers' pieces on the database. Users can share the post to up to 51 social media. All repertoire pages come with a comment function, users can comment down the page.
4. An instrument page that allows users to view repertoires by instrument.
5. A search function that allows users to search contents of the website, it also comes with some filters.
6. A forum page that allows users to join Corelia's group to interact with each other, the page also displays contents from selected composers and official accounts' social media like Facebook, Twitter, Tiktok, Youtube, users can comment, share, like or collect those posts to their own social media accounts.
7. A recommendation page displays some recommendation articles about composers, their works and other information.
8. A user authentication function that allows admin to login to the admin interface.
9. An admin interface that allows users to manage composers, their works, recommendation contents, an admin can add, edit, delete or search certain contents.
10. An about page to display the Corelia project's story and contact methods.

5. Implementation

Our team provided 4 different designs to the Corelia project, and let clients decide which design to proceed or any feature to be included. At the end, clients wished we could implement the design mentioned above on specifications. The innovative features to be included are search functions and forum functions. The design was implemented using Figma.

6. Processes(Hardware and Software)

To bring the project into development, there are a few processes:

1. Decide the development environment: all team members use the Window system, so development will be based on the Window system.
2. Decide technology stack: the project required a NoSQL as database service, MERN stack is the most popular technology stack for using NoSQL, M stands for MongoDB, E is Express.js, R is React, and N is Node.js. React is for client side development, it comes with built-in Bootstrap that helps React manage its styles. The MERN stack separates frontend and backend development by using tools like Postman for testing the portals(api) that connect them. It increases efficiency and reduces

maintenance cost. Every team member agrees on the stack because it's one of the most popular skill sets required in Australian IT companies.

3. Project plan and task allocation: a detailed project plan including project scope and exclusion, project deliverables, project timeline, goal and milestone, team breakdown and task allocation has been created, refer to research report.
4. Create Github repositories for the whole team to share codes and control versions.
5. Software to use: I used IntelliJ IDEA, Visual Studio Code and Sublime as code editor, Postman as API testing tools, Chrome as browser, Figma as prototype building tool.
6. Set up development environment: for this project, download Node.js 16.0.0 and its corresponding version of npm as package and server environment,

3. Individual Contribution

In this project, my contribution can be summarised and categorised as design, project planning, development, testing, team communication and team presentation.
For the design phase, I developed 1 design as shown in Fig.3.1, Fig.3.2 and Fig.3.3.
For the project planning phase, I actively participated in discussion with my team to decide the technology stack, features to be applied for the project, timeline of the project, task allocation, etc.
For the development phase, I work as a frontend team member together with Natalie Huang, the backend team are Mona Ma and Yichen Song. Before the development get started, I set up Github and tested the file structure for the team, I did some research on how MERN stack works and told other team member how frontend and backend work independently and how to use Postman for testing API that connects frontend and backend, that saved a lot of time for other team members. For the coding part, I added homepage, forum page, composer page, repertoire page, user authentication page,and about page to the project, more details can be referred to below description and technical logs.
For the testing phase, I worked with Natalie Huang to carefully combine our codes since we worked on different pages and functions, there were some conflicts on the code and we worked together to do debugging. It happened a couple of times.
For the presentation, I participated in the presentation with all other team members, including preparing presentation slides, introducing and displaying our final project website to clients.
For team communication, I have been communicating with other team members all the time, I never missed one team meeting or one discussion. I am always ready to listen to others' opinions and acknowledge their works. I believe this is how we build a strong team to motivate and encourage everyone to pursue the best outcomes of the project.

1. Results/Design prototype and explanations

1. Design prototype

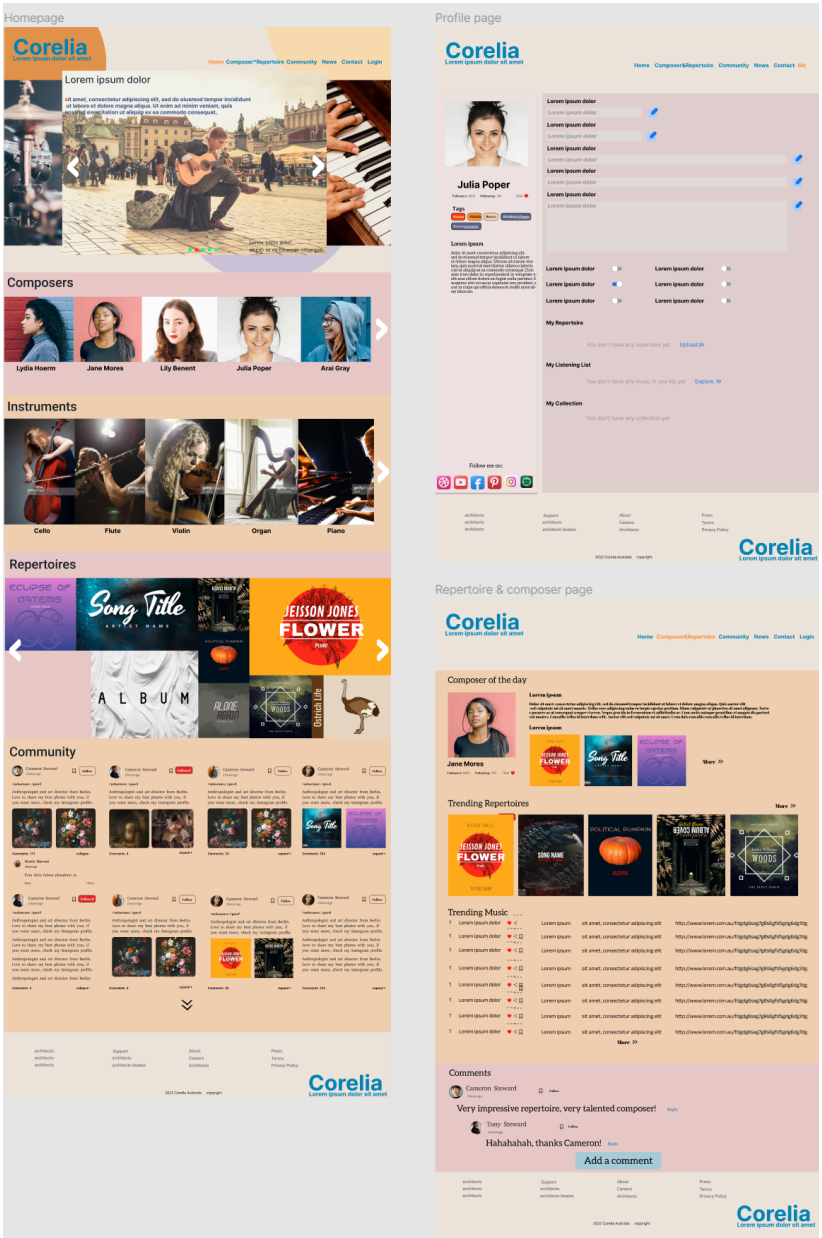


Fig.3.1 Homepage(left), composer page(top right), repertoire page(down right)

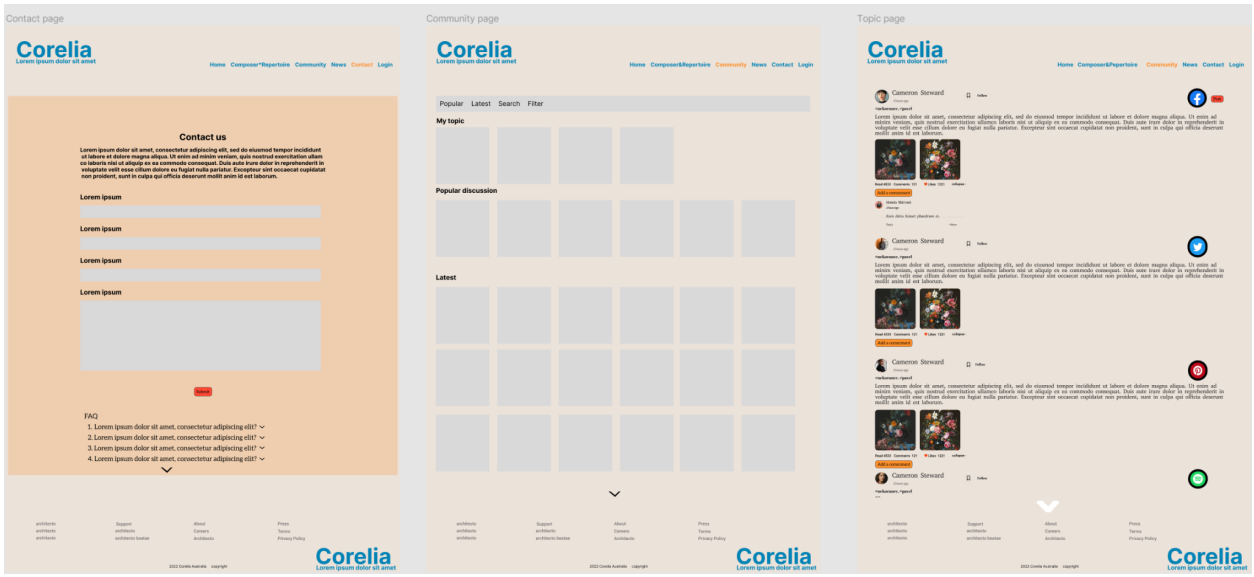


Fig.3.2 Contact page, community page and topic page

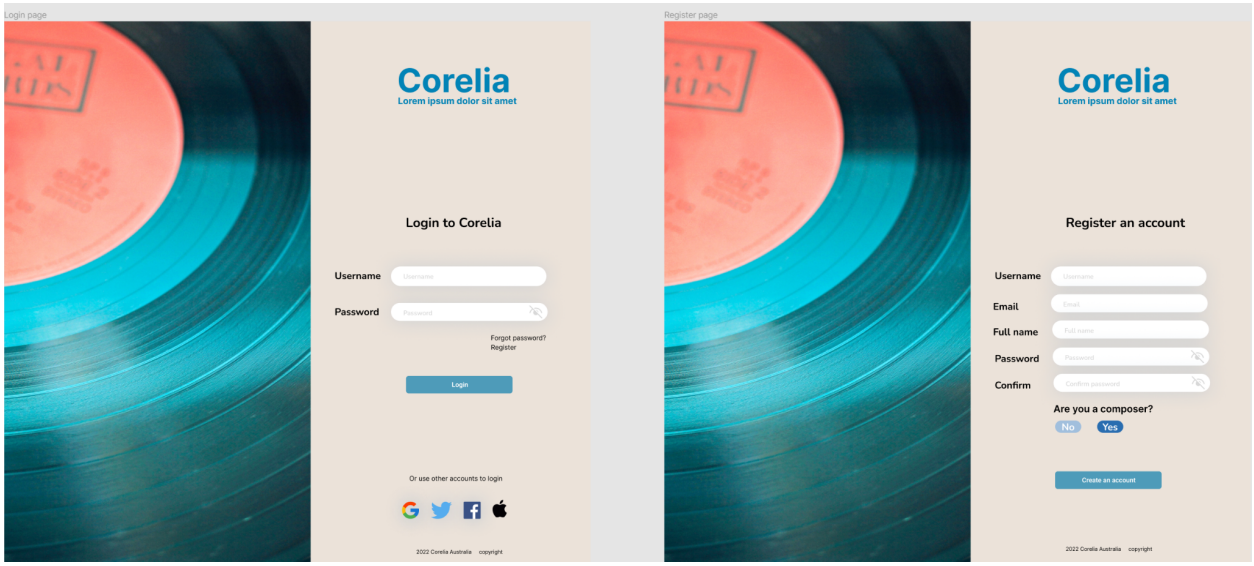


Fig.3.3 User login and register page

The above design was the version before client meeting. In Fig.3.1, the left side is the homepage. My idea is to use a slider on the top to display some upmost information, then display selected composers, their works, instrument categories and community posts in a smaller slider. The homepage is compact and provides comprehensive information about the website. Users can have a glimpse of what this website is about in seconds. It's also easy for users to navigate to another page. And then on the top right, the composer's profile page displays everything about the composer. On the down right is another version of the composer profile page, I was thinking of placing all composer's works under the composer's

profile, including some album and pieces inside the album, users can share, comment and like composer's pieces.
In Fig.3.2, the left side is a contact page, which I replaced with the About page in development. In the middle of Fig.3.2 is the community page. I assume that the page will display some selected accounts' content from different social media. Users can browse these content without having to open these social media websites or apps. The Corelia project and some composers might have their own social media accounts, it would be a good opportunity for them to show their feature content here and increase their influences. The right side of fig.2 is the forum page, I assume that the page will display some latest post/discussion from a selected social media group, everyone with a relevant account can join the discussion. The feature brings all relevant posts to one place, and it also utilises third-party's mature and stable product as discussion platform, it increases the interaction efficiency and reduces development cost.
Fig.3.3 is user authentication pages including login page and register page. Apart from regular login, the design includes third-party authentication, which makes it easier for users to login.

2. Setup team environment

I'm the first one to start coding in the team, because I wanted to find a proper way for everyone to work with MERN stack. Firstly, I set up the version control tools for the team, I created the repositories and added every team member as collaborator. Secondly, I did some research on how MERN stack works, and found frontend and backend work independently in 2 folders, client side folder and server side folder, I initiated the project with React.js and Node.js, and created a MondoDB Atlas database to verified how frontend and backend work independently. When I successfully retrieve JSON data from the route I wrote in server files by using Postman, I told other team members how to use Postman to test API and how the frontend is separated from backend in MERN stack. And for frontend, before the backend has done any data response, we use a dummy API service from https://jsonplaceholder.typicode.com/ to return JSON data for us to render web pages. In my past few project, frontend and backend were tied closely, that caused issue when each side were not at the same pace, one side will push other side, even worst, the faster side will blame other side for slower pace in regardless of different learning curve of technologies. What I did above was to find a pattern for frontend and backend to work efficiently and reduce conflicts. Lastly, I set up the frontend file structures including files to place router, styles, components, pages, images, etc. The structure reduces the frontend's task conflicts as each of us create or reuse a component and add it to our own

pages, the only files that we need to code together is app.js, which store the whole website's routers.

```
C:.
|   .gitignore
|   package-lock.json
|   package.json
|
+---public
|   \---image
|
+---src
|   |   App.js
|   |   App.scss
|   |   index.js
|   |
|   +---components
|   |   \---pages
|   |       +---AboutUs
|   |       +---Admin
|   |       +---Authentication
|   |       +---Composer
|   |       +---Forum
|   |       +---Home
|   |       +---Instrument
|   |       +---Navbar
|   |       +---Profile
|   |       +---Recommendation
|   |       +---Repertoire
|   |       +---Search
|   |
|   \---img
```

Fig.3.4 File structure

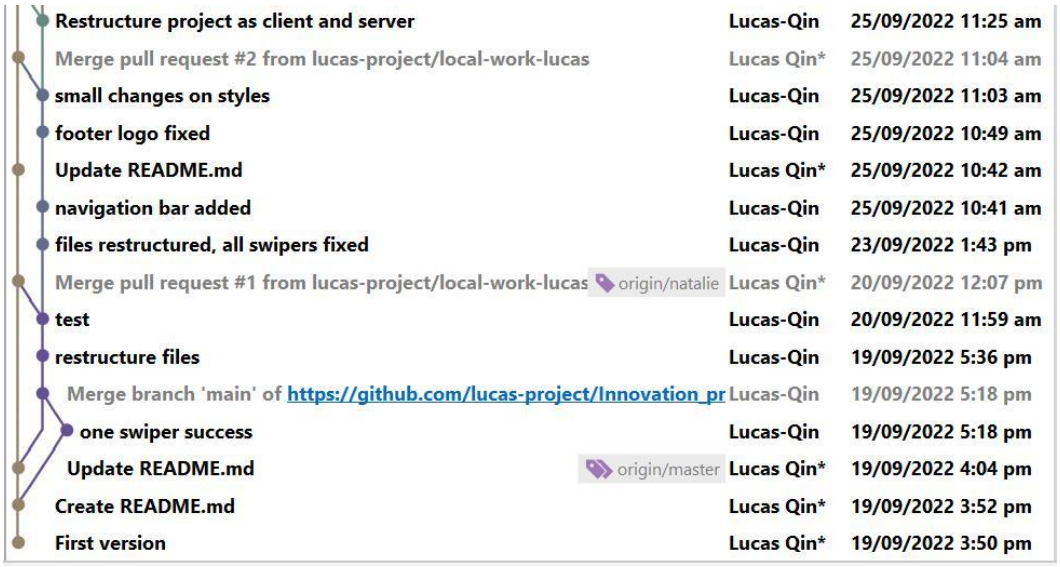


Fig.3.5 Technical log for initialise project structure

3. Add homepage

From here, the team stepped into the development phase of the project. The homepage basically inherits the prototype design as shown in Fig.3.1, which consist of a brief navigation bar, a big picture with some catchy titles, sliders for top repertoire, popular composer and top-searched instrument, down the page is a footer with comprehensive information of the website.

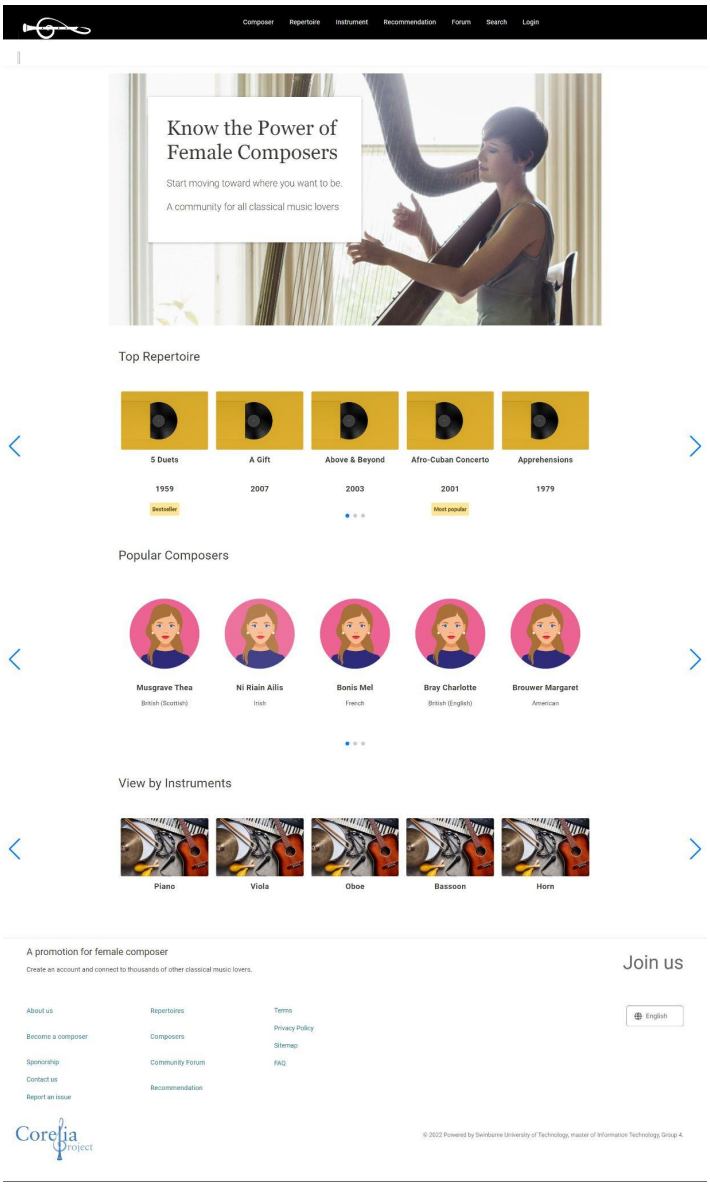


Fig.3.6 Final homepage

4. Add composer page

The composer page consists of two levels of directories. The first level of directory are all the composers in the database as shown in Fig.3.7, which only come with a profile picture, a name and nationality. Due to the reason that our client did not provide us all the profile pictures for the composer, here I used a placeholder pic for every composer.

If you click on one of the composers, the website will be redirected to the profile page of the composer as shown in Fig.3.8. The detailed profile page displayed all the information provided by clients, including their name, date of birth, website, etc. A button “view her pieces” under the Biography section also provided for users to find out her works. The page integrated with a share function, which enabled users to share this composer to up to 50 websites or portals as shown in Fig.3.11.

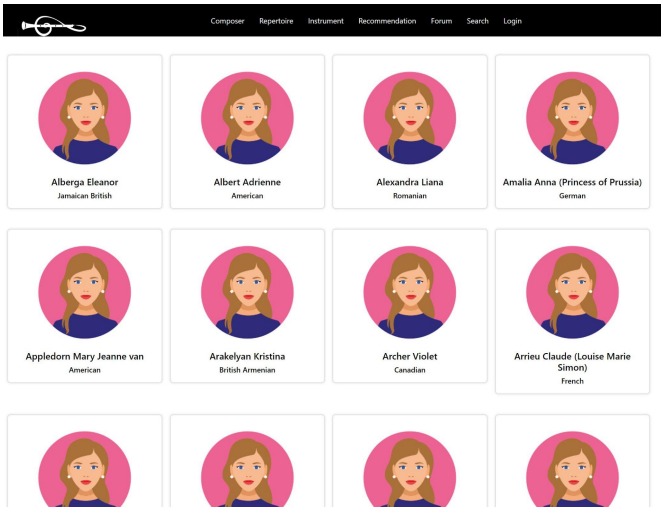


Fig.3.7 Final composer page

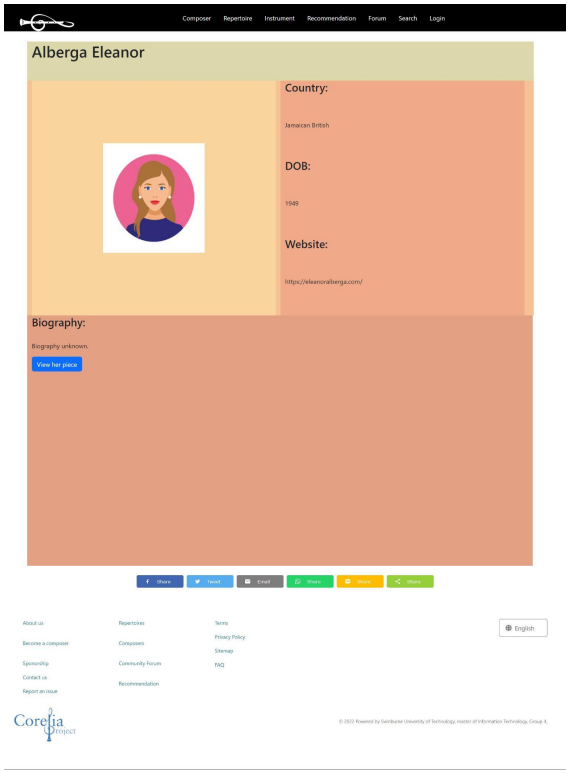


Fig.3.8 Final composer's profile page

5. Add repertoire page

As the Corelia project aims to build the website as a centralised database for all composers and their works, I added an extra page for users to explore and discover all the works in one place. All the works will be sorted in an alphabetical order. If you click on a piece, the website will be redirected to the details page of the piece, the page will be displayed with all the information about the work. As a composer's profile page, users can also share the page anywhere they want. There's an innovative feature in the page, I added a comment function for the page so that everyone can comment down below the piece. The comment function is from Facebook's comment API, the API provides comments to all the public websites and stores the data in Facebook's own database. The API reduces the development cost as it is integrated services and it is a free service. It also increase the website's stability and development efficiency as it provides a mature function with constantly update and maintenance.

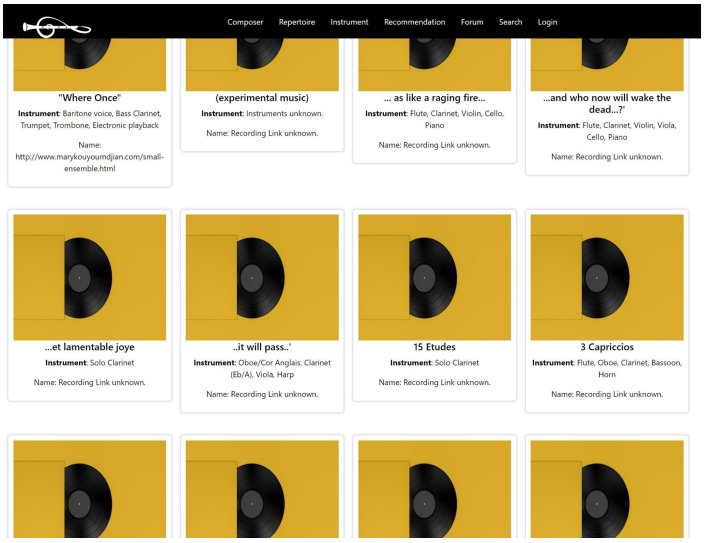


Fig.3.9 Final repertoire page

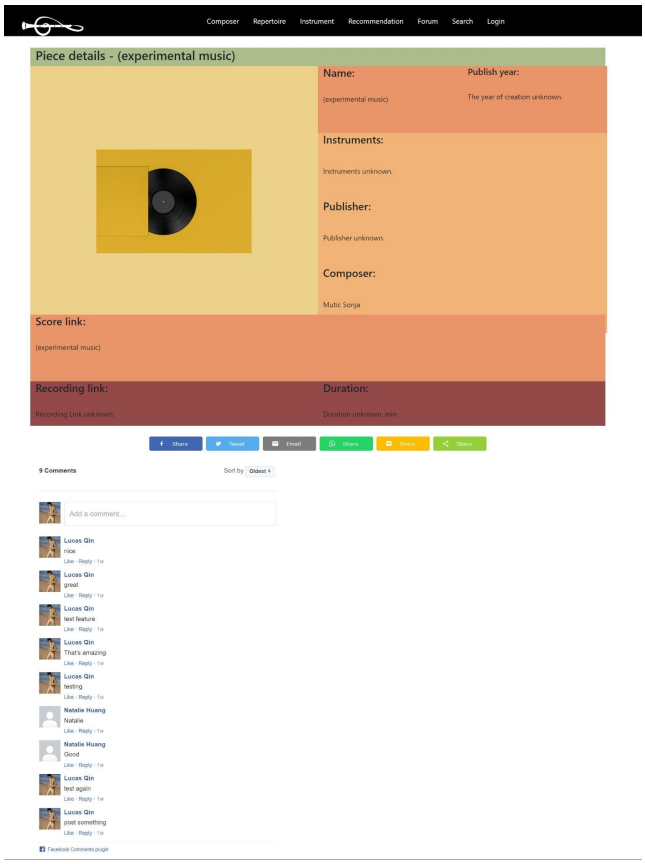


Fig.3.10 Final repertoire detail page

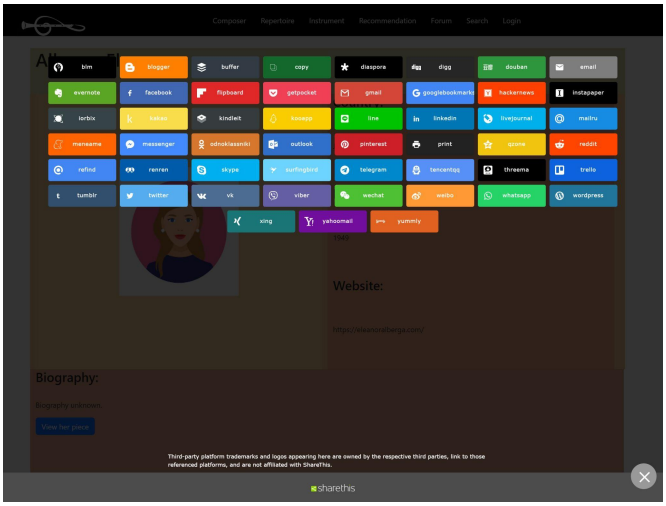


Fig.3.11 Share function integration

6. Add forum page

The forum page is one of the innovative features I added to the project. Before the project, I had 2 solutions for the building community for the Corelia project. One is to build a forum from scratch, the forum page should have functions like posting new discussions, reply, comment, like, share, category the topic, follow, management, security and many others. To achieve these functions without glitch could take a very long time and with large amounts of investment. Another is to use mature and stable third-party service as the website's own community function. These third-party API usually have a completed and stable product that has been tested by the market, their functions are more user-oriented and up-to-date since most of them are built and maintained by a professional team for years. Connecting the website with these API could significantly reduce development cost and risk. The only thing that needs to worry about is how those services charge. Luckily at the moment, I found many companies like Facebook, Twitter, Tiktok, and Google, they provided thorough documents and free service for websites to use their functions.

The final choice was the last one, to build a website community with third-party functions. As shown in left of the Fig.3.12, the first service I integrated is the Facebook's group function, this function allow user to create a group in Facebook and display it on their website, including cover picture, title, and a join group link, if user has a Facebook account logged in their computer, the link will add them to the group. Then users can interact with all the other members and what's more, the group will additionally attract classical music lovers from Facebook, making it an even bigger group. The website user now can enjoy all the functions provided by Facebook to interact with each other, they can post here, comment on a post, share or like it, or follow a member, or block a member.

Then I added contents from Facebook, Tiktok, Twitter and Youtube, all the contents are displayed in sliders, users can swipe to find more contents from the same platform. Admin can choose those contents from composer's account, their official account, or cooperated business accounts, all media can play the website without having to redirect to its original website, making it an excellent opportunity to promote composers or their works regularly.
In the bottom of the page as shown in the right side of Fig.3.12, I used Twitter's timeline API to display all the public posts of the official account, these contents are dynamic, meaning whenever the account has an update on Twitter, it will also show here simultaneously. That would be a great way for the Corelia project to display their account or announce news here to every user. In the future, these pages can be integrated with more functions and contents from more other social media.

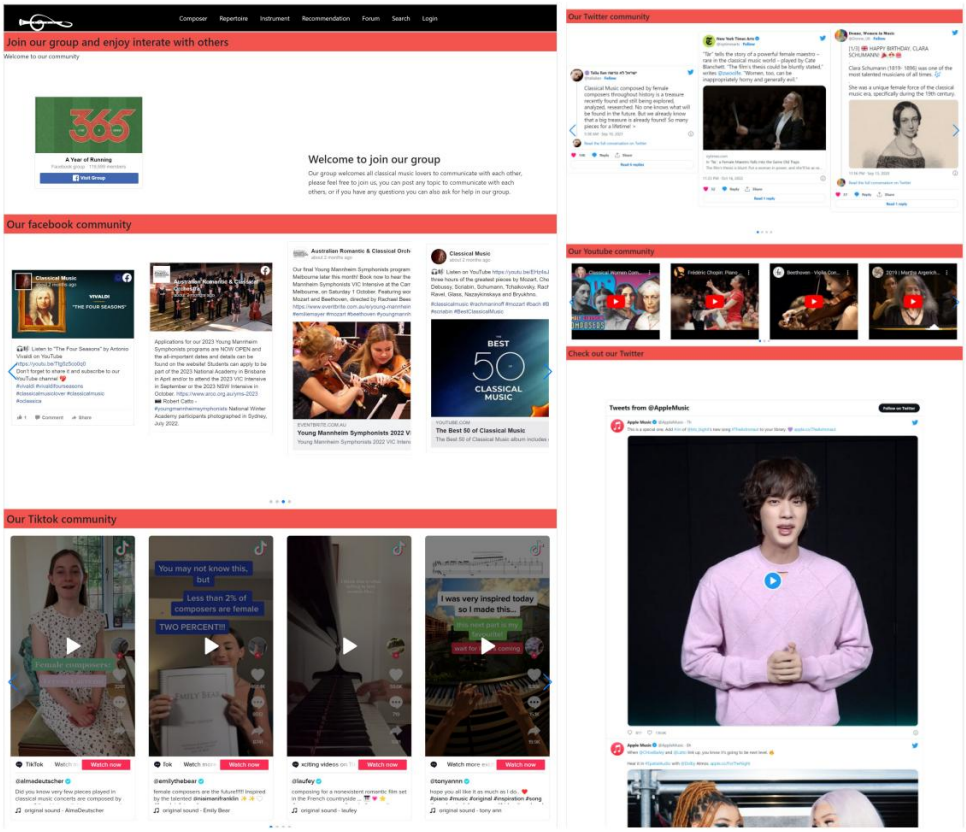


Fig.3.12 Final forum page(left is top the page, right is the image when scroll the page down)

7. Add user authentication

This project's user authentication as mentioned by clients was not a prioritised feature they wanted, hence we tried to make it as simple as it is to only use it

when absolutely necessary. In our project, the only reason to use user authentication is when an admin needs to update the contents of the website, for example, adding a new composer or editing a repertoire, etc.
An admin needs to provide their email and password to register an account, then type in correct information to login, if successful, the page will be redirected to the admin page.

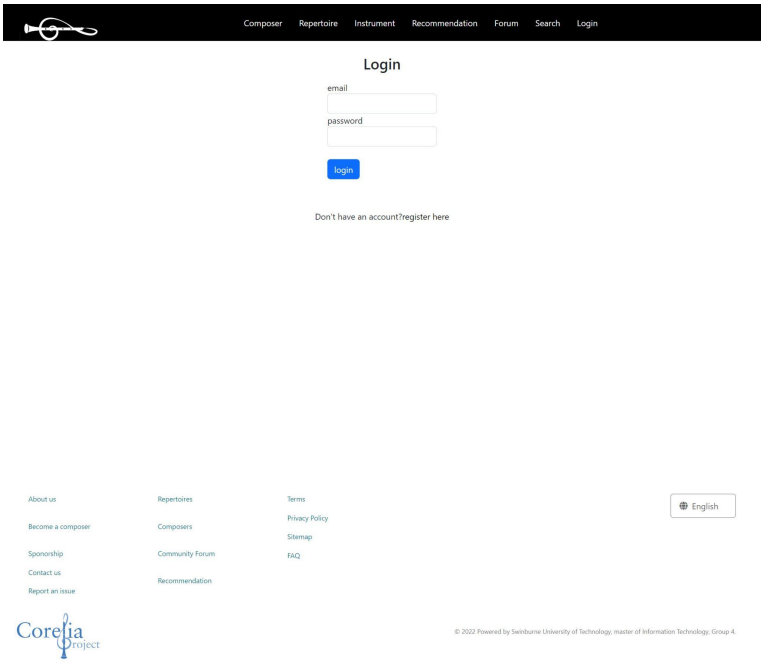


Fig.3.13 Final login interface

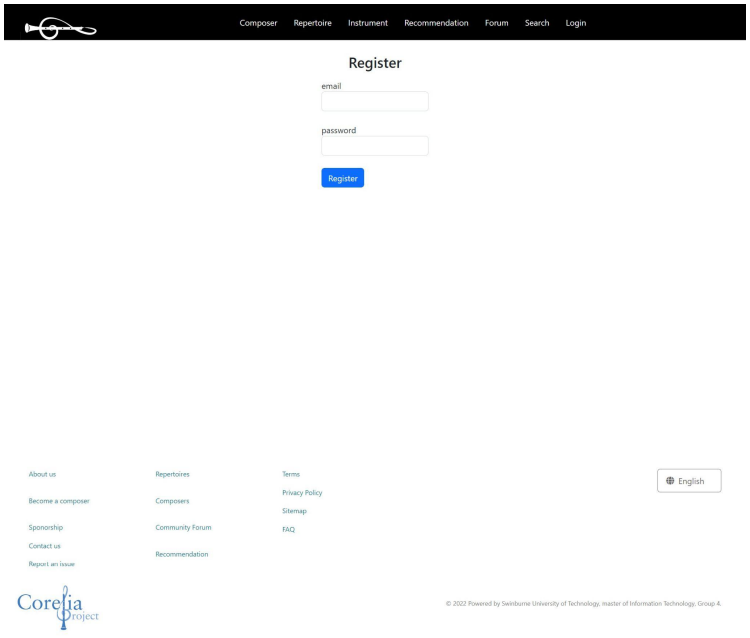


Fig.3.14 Final register page

8. Add about page

The about page displays some basic information of the Corelia project.

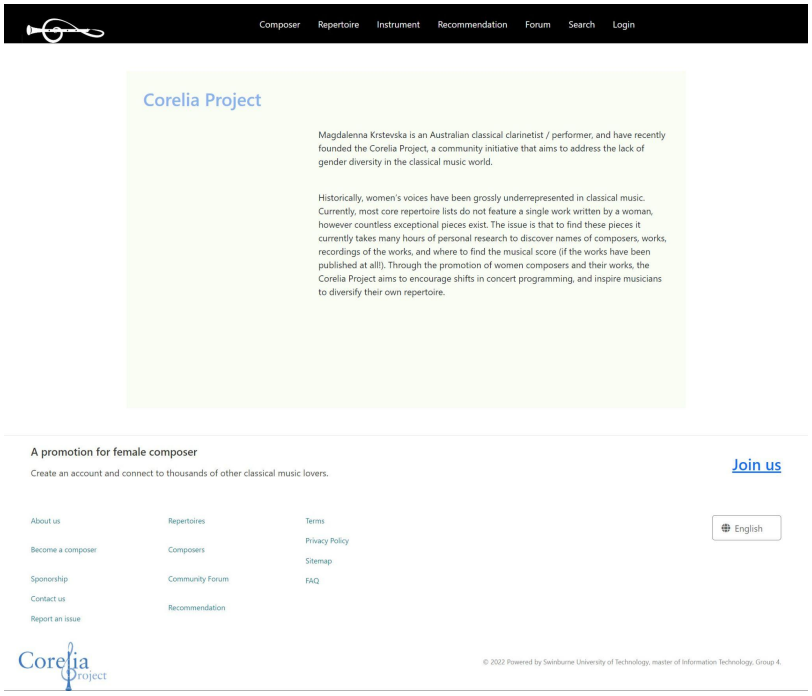


Fig.3.15 Final about page

2. Project outcomes(evaluates results/justification of design)

The project outcomes mentioned above satisfied clients requirement, the reason are as below:

1. Clients wished to include a full database list of works and comprehensive information provided by them. We have a composer page that displays all composers information, repertoire page that displays all works in the database, instrument page that allows users to view composer's works by particular instruments, search function for searching any particular contents in the database.
2. Clients want to show the composer's profile, we have a composer profile page that shows all provided information as in Fig.3.8.
3. Clients need a blog on the website to display some featured works or recommended contents. We have a recommendation page that does the exact job.

4. Clients want a community forum page for members to interact with each other, plus commenting and sharing. I have added a comment function to all the composer's works and composer's profile page. There is also a forum page that enables users to join an official forum empowered by Facebook, they can post a discussion, comment on others posts, like, share or collect a post or follow a person in the group. They can also find selected contents from different composers' accounts in Facebook, Twitter, Tiktok and Youtube, all the media can be played straightly on the website.
5. Clients want a NoSQL database, we used MongoDB as a database service.
6. Clients added a search function requirement after meeting, we have provided a powerful search function that comes with a complex filter.

4. Reflections

1. Contribution and how I met learning outcomes

To get started, I have summarised my contribution as the following. Firstly, I participated in the whole project without missing any meeting or discussion. I helped the team to plan the project's timeline, task allocation, deliverables, goals, technology stack, etc. I also contribute a design in conceptual design. Secondly, I helped the whole team set up the development environment and showed them hand-by-hand how it works. Thirdly, I work as one of two frontend team members and added homepage, composer page, repertoire page, forum page, user authentication function, comment and share functions, and about page to the project independently. I also worked with other team members to reduce code conflict, for example the route definition, the package version, etc. Fourthly, I helped the team in testing the whole website every time there is a code merge operation. Fifthly, I provided suggestions to other team members in terms of solutions, technologies to be used, and I listened to their opinions on the design, and the codes. I believe a good team is achieved by good communication, and listening and acknowledging of other's works are very important parts of it. Lastly, I helped the team to prepare for the final presentation, making sure all the works were met with requirements.

Except for the good parts, I also found there are some improvements I can make in the next project. I was behind schedule most of the time due to many reasons, but I did not speak to other team members about it. I will communicate more with others next time and seek support.

Now I will talk about how I met learning outcomes. First of all, I learned how to apply an evidence-based approach to innovation. From the very beginning of the project, I did a lot of research on innovative design, I read and studied at least 5 academic articles and used a Design Sprint Methodology to do conceptual design, more details refer to assignment Conceptual design. Secondly, I applied real-world research practices through case study in literature reviews, the result can be referred to my assignment Research Report. I plan, design, and develop innovative solutions through cooperation with my team members. We carefully evaluate and identify the pros and cons of the method or ideas from literature review and come to agreements on innovative ideas through multiple discussions. Thirdly, I learned how to work and communicate with clients, the description of the project in Canvas was missing a lot of details, hence we were not sure how our clients' business was operated, and what are the features that they valued most under current circumstances. Through a questioning by tutor and the meeting with clients, we know that apart from the feature listed before, our clients consider search function a must for the project website. Fifthly, the project is a MERN stack project, that means the project uses React.js and Bootstrap as frontend, Node.js, Express.js and MongoDB as backend. I learned the technology stack through this project, now I'm confident enough to apply for any junior job using these technologies. Lastly, I have learned how to work as a team to generate a solution. A good communication is the good start of team corporation, I always listen to other team members opinions and acknowledge their works, I will not save my words on complimenting everyone if they did a good job, I think acknowledgement of others job reduce potential conflict and increase everyone morale, it motivates team members to work toward a better outcome.

2. Examples for learning outcomes

Example 1: I have applied an evidence-based approach to my design. In my concept design(refer to assignment conceptual design). I followed the Design Sprint Methodology developed by Google to generate 3 different designs, and through careful justification and Vulnerability analysis, I selected one design as my final design.

Example 2: I have learned how to apply React.js and Bootstrap to projects, it is my technical learning outcome. I have learned how to initialise the project, set up the environment, corporate with team members with different task within MERN stack, test API with Postman, and eventually, I contributed a homepage, a composer page, a repertoire page, a community forum page, an about page, and an user authentication function, without learning how to code with React.js, I

couldn’t be able to do the contribution, so this can be considered as the example of my technical learning outcome.

3. Technical learning

Our team member decided to use the MERN stack for the project. I worked as a frontend team member as mentioned above. The technologies I learned and used for frontend are React.js, Bootstrap, Postman and third-party APIs.

React.js

React.js is the main technology that renders client interface, React.js is a Javascript framework and library for building interactive and dynamic user interface, it is currently maintained by Facebook. One of the advantages React offers is that it allows developers to build many reusable components. When rendering a page, simply import the component without having to write code repeatedly. Fig.4.7 shows the components I created for the project. Take the website footer as an example. Footer can be displayed in many pages, in order to avoid repeat code, I made it as a reusable component. The process can be like:

1. Import necessary React components

```
import styled from "styled-components";
import LogoImg from "../img/Corelia Project_Master Logo_Colour_PNG.png";
import {Link} from "react-router-dom";
```

Fig.4.1 import components

2. Render the page

```
const Footer = () => {
  return (
    <StyledFooter>
      <div className='footer-links'>
        <div className='links-left'>
          <div className='main-links'>
            <Link
                to={{pathname:'/about'}}
            ><p>About us</p></Link>
            <Link
                to={{pathname:'/api/users'}}
            ><p>Become a composer</p></Link>
            <a href='#'>Sponorship</a>
            <a href='#'>Contact us</a>
            <a href='#'>Report an issue</a>
          </div>
          <div className='support-links'>
            <Link
                to={{pathname:'/pieces'}}
            ><p>Repertoires</p></Link>
            <Link
                to={{pathname:'/api/composers'}}
            ><p>Composers</p></Link>
            <Link
                to={{pathname:'/forum'}}
            ><p>Community Forum</p></Link>
            <Link
                to={{pathname:'/recommendation'}}
            ><p>Recommendation</p></Link>
          </div>
          <div className='terms-links'>
            <a href='#'>Terms</a>
            <a href='#'>Privacy Policy</a>
            <a href='#'>Sitemap</a>
            <a href='#'>FAQ</a>
          </div>
        </div>
        <div className='links-right'>
          <a href='#' className='language-link'>
            <i class='fas fa-globe'></i> English
          </a>
        </div>
      </div>

      <div className='copy-right'>
        <img src={LogoImg} alt='Logo' className='logo' />
        <p>&copy; 2022 Powered by Swinburne University of Technology, master of Information Technology, Group 4.</p>
      </div>
    </StyledFooter>
  );
};
```

Fig.4.2 render page

3. Style the page
   Fig.4.3 used SCSS for rendering the component, SCSS is powerful than CSS, it enable more efficient syntax of CSS, for example nested element can be placed inside its parent element's style curly bracket, but in CSS, the child style must at the same level as parent style, making the structure hard to maintain when it become too long. Fig.4.3 only show very little part of the code as an example, more code can be found in the Github link.

```scss
const StyledFooter = styled.div`
  display: flex;
  flex-direction: column;
  .footer-links {
    width: 100%;
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
    flex-wrap: wrap;
    padding: 1rem 3rem;
    margin-top: 1rem;
  }

  .links-left {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 8rem;

    a {
      display: block;
      text-decoration: none;
      padding-bottom: 1rem;
      color: #0f7c90;
      font-size: 0.82rem;

      &:hover {
        color: #004450;
      }
    }
  }
```

Fig.4.3 Add SCSS style

4. Export the component

```
export default Footer;
```

Fig.4.4 Export component

5. Import the component to the page that need the component

```
import Footer from "../../Footer";
```

Fig.4.5 Import component

6. Render the component in the page

```
<footer>
  <HomeFooterJoinUs />
  <Footer />
</footer>
```

Fig.4.6 Render component

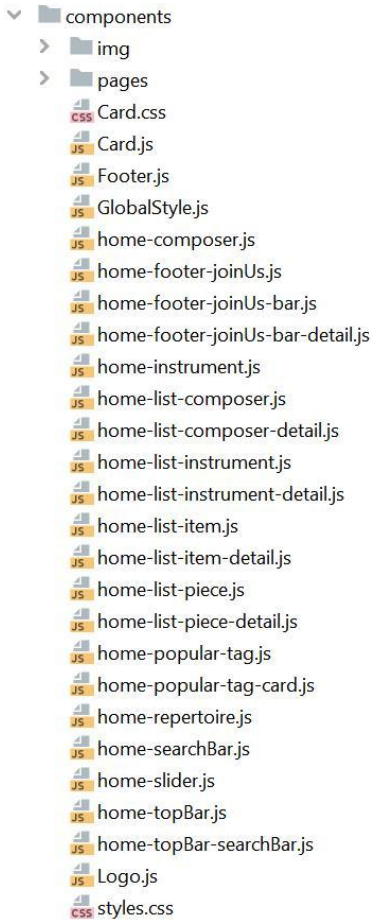


Fig.4.7 Components I added

The key point of React is that it needs to get data from the backend and render those data into specific pages. The basic logic is the frontend defines a route to send requests, and uses AXIOS or React's Fetch() method to send requests to a route that was defined on the backend. Next the backend writes some codes like if you receive a certain type of request from a pathname, then return certain data queried from MongoDB in a JSON format. When the frontend receives the JSON data, decode it and get its keys and values, then render it in some pages. Since there are too many routes, here I take composer page as an example to explain the process:

1. In App.js, import React's Routes and Route components.

```
import { Routes, Route } from "react-router-dom";
```

Fig.4.8 Import React components

2. Import composer page file.

```
import Composer from "./components/pages/Composer/Composer";
```

Fig.4.9 Import composer file

3. Render Routes and Route components.

```
<Routes>
  <Route element={<WithoutNav />}...>
  <Route  element={<WithNav />}...>
</Routes>
```

Fig.4.10 Add router

4. Add composer page file to router, meaning URL ending with "/api/composers" will be handled and rendered by composer file in browser.

```
<Route path="/api/composers" element={<Composer />} />
```

Fig.4.11 Add composer file to router

5. Then in the composer page, import the necessary components prepared for rendering.

```
import React, { useState, useEffect } from "react";
import { Link } from "react-router-dom";
import Card from "../../Card";
import "./Composer.css";
```

Fig.4.12 Import necessary components

6. Use React's built-in hooks useState() for later store fetch results.

```
const [albums, setAlbums] = useState( initialState: []);
```

Fig.4.13 Declare component

7. Use React's built-in hook useEffect() for sending http requests to the backend and handle JSON data returned with useState(). The useEffect() stores JSON data into an empty array at the end.

```
useEffect( effect: () => {
    setIsLoading( value: true);
    const fetchData = async () => {
        const result = await fetch( input: "http://localhost:3000/api/composers");
        const resultJson = await result.json();
        // const sliceResult = resultJson.slice(0, 10);
        setAlbums(resultJson);
        setIsLoading( value: false);
    };
    fetchData();
}, deps: []);
```

Fig.4.14 Fetch() method

8. Render JSON data by using Javascript's map() method since the data has been stored in an array in the last step.

```
return (
    <div>
        {isLoading && (
            <div className="loading">
                <p>...loading</p>
            </div>
        )}

        <div className="albums-container">
            {albums.map(album => (
                <Link
                    to={{pathname:`/api/composers/${album._id}`}}
                    state={{
                        _id:album._id,
                        name: album.name,
                        nationality:album.nationality,
                        DOB:album.DOB,
                        website:album.website,
                        biography:album.biography,
                        image:album.image

                    }}
                    key={album.name}
                    style={{ textDecoration: "none", color: "black" }}
                >
                    <Card className="albums-card">
                        <div>
                        <img
                            // src={"https://via.placeholder.com/168x118.png"}
                            // src={album.image}
                            src="/image/placeholder.png"
                            alt={`data thumbnail`}
                            style={imgStyle}
                        />
                        <h5>{album.name}</h5>
                        <h6>{album.nationality}</h6>
                        </div>
                    </Card>
                </Link>
            ))}
        </div>
    </div>
);
```

Fig.4.15 Render data returned by backend

8. Step 7 is to render a Card component for every item in JSON. Here I want to make the Card component clickable, so I used React's built-in Link component. The Link component has some attributes inside, the "to" contribute is the URL that the card will be redirected to, the URL must be predefined in App.js as in step one. The "state" attribute can pass any parameters to the page that is jumped to, here I pass composer's information dynamically to the next page. The "key" attribute ensures that every Link is unique. The "style" attribute can add some basic styles to components inside Link. These parameters can be retrieved using React's hook useLocation().

```
<Link
    to={{pathname:`/api/composers/${album._id}`}}
    state={{
        _id:album._id,
        name: album.name,
        nationality:album.nationality,
        DOB:album.DOB,
        website:album.website,
        biography:album.biography,
        image:album.image

    }}
    key={album.name}
    style={{ textDecoration: "none", color: "black" }}
>
    <Card className="albums-card"...>
</Link>
```

Fig.4.16 Add link to each component

Postman

I learned how to use Postman for testing API. In MERN stack, the backend and frontend are connected with API, API normally return or receive JSON/Array format data according to request URL, request type, port number and hostname. In fact, Postman can be applied to many other technology stacks since web requests and data transition using JSON are also applied in those technologies.
Postman is the key point to keep frontend and backend development separated. Take Fig.4.17 as an example, I sent a request from frontend, method is “GET”, URL is “http://localhost:3000/api/composers”, the backend then returned to me all composer’s information. However, if I send a “POST” request as Fig.4.18, there will be a 404 error, meaning no such request handler found on the backend. In this way, I don’t need to know how backend codes look like, or what library or tools were used, I only care about the data returned by the backend, as long as I have the data I need, I don’t need to care how these data query from the database. But if there’s no data returned, either if the frontend’s request goes wrong or backend’s handler goes wrong, if the request has no problem, I then can talk to the backend team and ask them to check this particular handler.

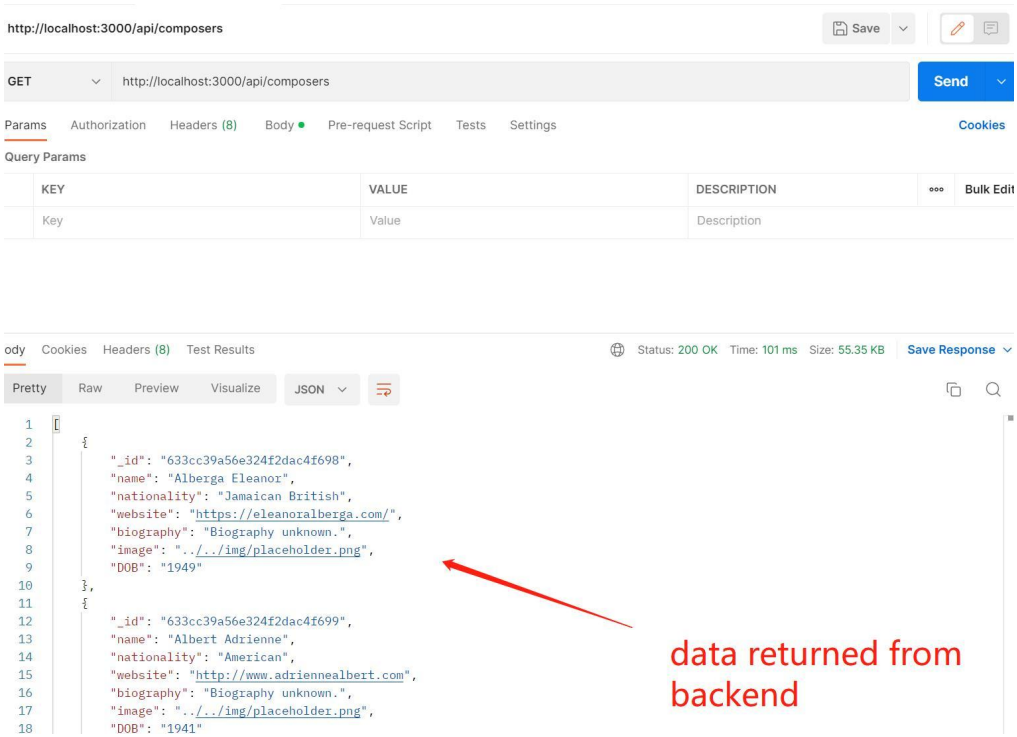


Fig.4.17 Successful request

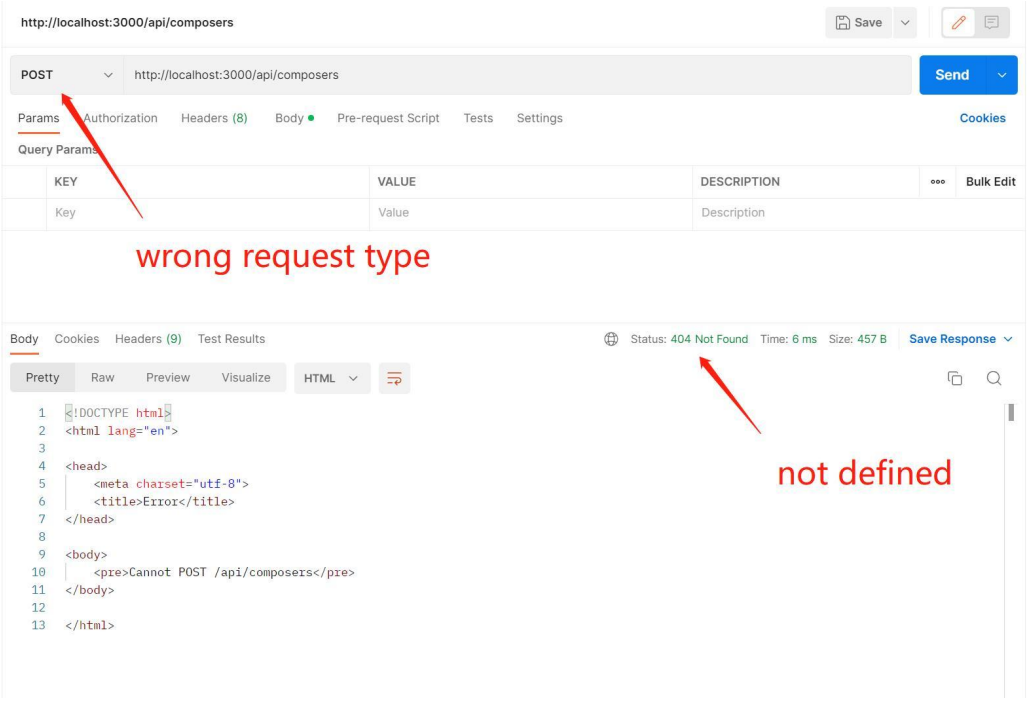


Fig.4.18 Unsuccessful request

5. Conclusion and Future Recommendations/Implementation

In this article, I used evident-base approaches to create the conceptual design, to be specific, those approaches are literature review and Sprint Design methodology, which helped me to produce evident-support and practical innovative features. Then I talked about my contribution to the project, including design, team corporation, team communication, development, testing and presentation. I presented a lot of details of the page/functions I have contributed. Lastly, I added reflection to the project, which reflected my contribution and how I met the requirements of learning outcomes. I provided some examples to support my points.

Although the project has reached our team's expectations, due to constraints of time, cost, and experience, the website still has a lot of potential in terms of future implementation. My recommendation for future improvement are as below:

1. All placeholder pictures for example composer's profile page to be replaced by real pictures in the future.
2. The project hasn't gone online yet, it will need to host in a cloud server in the future.
3. Currently users can only share and comment on the composer's work, a collect function needs to be added in the future.
4. The project's style needs more work on device compatibility.
5. All composers and repertoire display in one page now, it needs to be split into many pages in the future.
6. The website's security needs more work as we haven't done much on security.
7. We don't have a non-composer user's profile now, it will need to be added in the future.

Word count: 5306

6. Reference

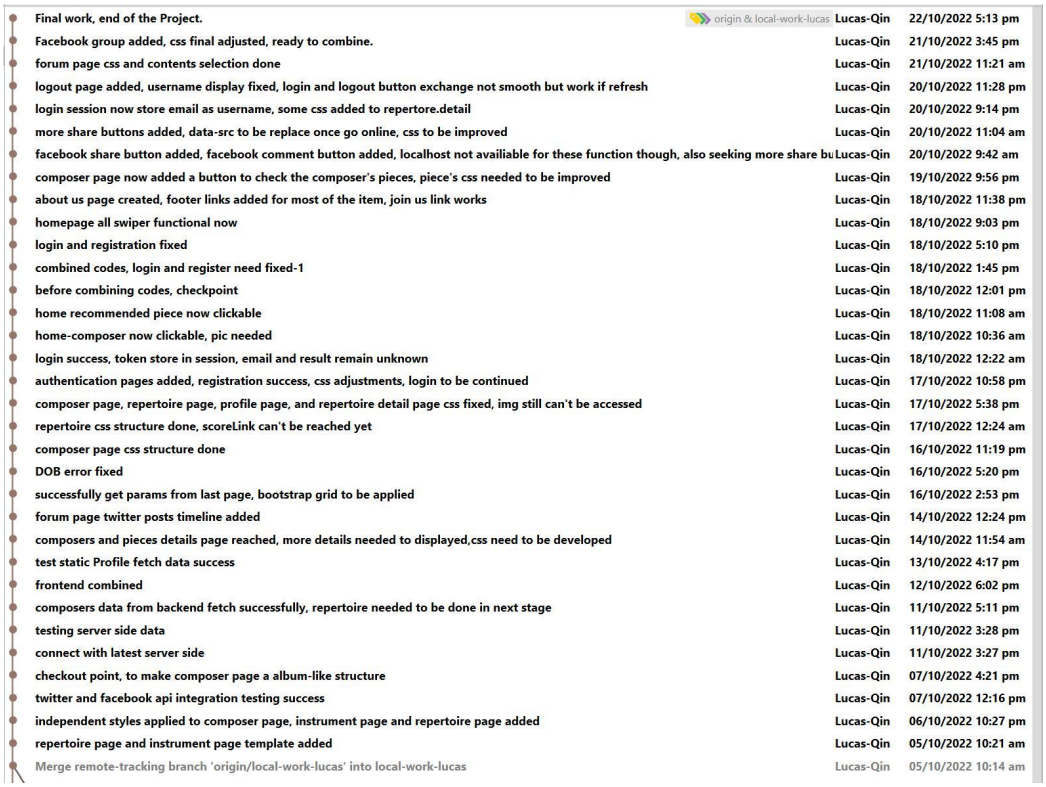
7. Appendices

1. Team work breakdown(Individual contribution)

Team member	Contribution
Lucas Qin	Conceptual design Set up development environments Composer pages Repertoire pages Homepage Community forum page About page User authentication function Comment function Share function Testing Presentation

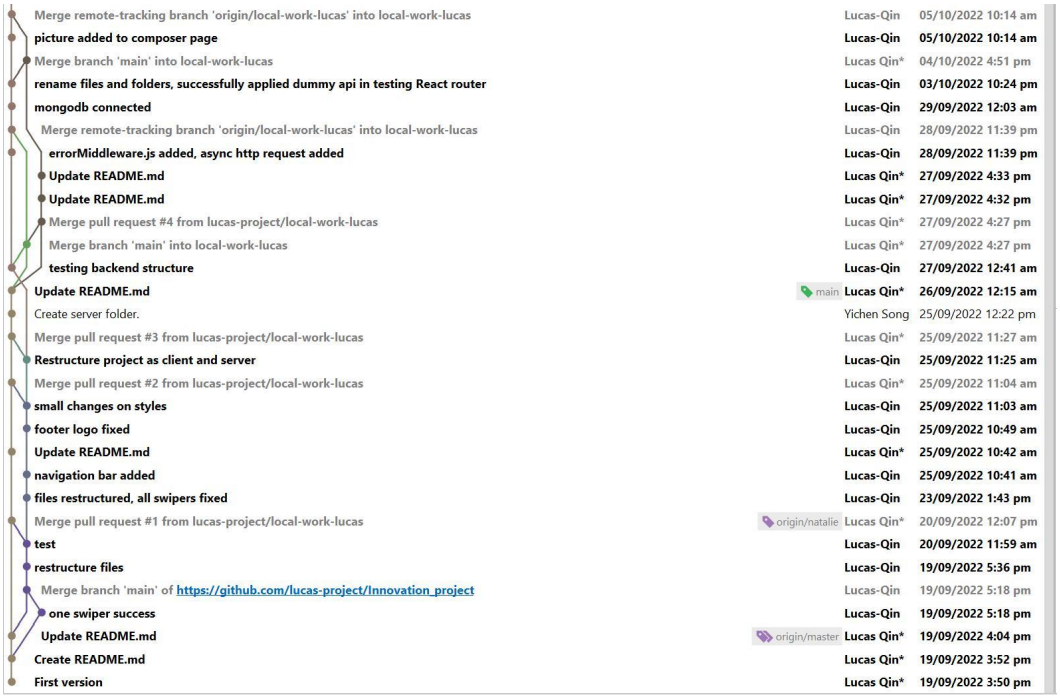
Table 1. Individual contribution

2. Technical log



Final work, end of the Project.	origin & local-work-lucas	Lucas-Qin	22/10/2022 5:13 pm
Facebook group added, css final adjusted, ready to combine.		Lucas-Qin	21/10/2022 3:45 pm
forum page css and contents selection done		Lucas-Qin	21/10/2022 11:21 am
logout page added, username display fixed, login and logout button exchange not smooth but work if refresh		Lucas-Qin	20/10/2022 11:28 pm
login session now store email as username, some css added to repertore.detail		Lucas-Qin	20/10/2022 9:14 pm
more share buttons added, data-src to be replace once go online, css to be improved		Lucas-Qin	20/10/2022 11:04 am
facebook share button added, facebook comment button added, localhost not availiable for these function though, also seeking more share bu		Lucas-Qin	20/10/2022 9:42 am
composer page now added a button to check the composer's pieces, piece's css needed to be improved		Lucas-Qin	19/10/2022 9:56 pm
about us page created, footer links added for most of the item, join us link works		Lucas-Qin	18/10/2022 11:38 pm
homepage all swiper functional now		Lucas-Qin	18/10/2022 9:03 pm
login and registration fixed		Lucas-Qin	18/10/2022 5:10 pm
combined codes, login and register need fixed-1		Lucas-Qin	18/10/2022 1:45 pm
before combining codes, checkpoint		Lucas-Qin	18/10/2022 12:01 pm
home recommended piece now clickable		Lucas-Qin	18/10/2022 11:08 am
home-composer now clickable, pic needed		Lucas-Qin	18/10/2022 10:36 am
login success, token store in session, email and result remain unknown		Lucas-Qin	18/10/2022 12:22 am
authentication pages added, registration success, css adjustments, login to be continued		Lucas-Qin	17/10/2022 10:58 pm
composer page, repertoire page, profile page, and repertoire detail page css fixed, img still can't be accessed		Lucas-Qin	17/10/2022 5:38 pm
repertoire css structure done, scoreLink can't be reached yet		Lucas-Qin	17/10/2022 12:24 am
composer page css structure done		Lucas-Qin	16/10/2022 11:19 pm
DOB error fixed		Lucas-Qin	16/10/2022 5:20 pm
successfully get params from last page, bootstrap grid to be applied		Lucas-Qin	16/10/2022 2:53 pm
forum page twitter posts timeline added		Lucas-Qin	14/10/2022 12:24 pm
composers and pieces details page reached, more details needed to displayed,css need to be developed		Lucas-Qin	14/10/2022 11:54 am
test static Profile fetch data success		Lucas-Qin	13/10/2022 4:17 pm
frontend combined		Lucas-Qin	12/10/2022 6:02 pm
composers data from backend fetch successfully, repertoire needed to be done in next stage		Lucas-Qin	11/10/2022 5:11 pm
testing server side data		Lucas-Qin	11/10/2022 3:28 pm
connect with latest server side		Lucas-Qin	11/10/2022 3:27 pm
checkout point, to make composer page a album-like structure		Lucas-Qin	07/10/2022 4:21 pm
twitter and facebook api integration testing success		Lucas-Qin	07/10/2022 12:16 pm
independent styles applied to composer page, instrument page and repertoire page added		Lucas-Qin	06/10/2022 10:27 pm
repertoire page and instrument page template added		Lucas-Qin	05/10/2022 10:21 am
Merge remote-tracking branch 'origin/local-work-lucas' into local-work-lucas		Lucas-Qin	05/10/2022 10:14 am

Fig.7.1 Technical log part-1



Merge remote-tracking branch 'origin/local-work-lucas' into local-work-lucas		Lucas-Qin	05/10/2022 10:14 am
picture added to composer page		Lucas-Qin	05/10/2022 10:14 am
Merge branch 'main' into local-work-lucas		Lucas Qin*	04/10/2022 4:51 pm
rename files and folders, successfully applied dummy api in testing React router		Lucas-Qin	03/10/2022 10:24 pm
mongodb connected		Lucas-Qin	29/09/2022 12:03 am
Merge remote-tracking branch 'origin/local-work-lucas' into local-work-lucas		Lucas-Qin	28/09/2022 11:39 pm
errorMiddleware.js added, async http request added		Lucas-Qin	28/09/2022 11:39 pm
Update README.md		Lucas Qin*	27/09/2022 4:33 pm
Update README.md		Lucas Qin*	27/09/2022 4:32 pm
Merge pull request #4 from lucas-project/local-work-lucas		Lucas Qin*	27/09/2022 4:27 pm
Merge branch 'main' into local-work-lucas		Lucas Qin*	27/09/2022 4:27 pm
testing backend structure		Lucas-Qin	27/09/2022 12:41 am
Update README.md	main	Lucas Qin*	26/09/2022 12:15 am
Create server folder.		Yichen Song	25/09/2022 12:22 pm
Merge pull request #3 from lucas-project/local-work-lucas		Lucas Qin*	25/09/2022 11:27 am
Restructure project as client and server		Lucas-Qin	25/09/2022 11:25 am
Merge pull request #2 from lucas-project/local-work-lucas		Lucas Qin*	25/09/2022 11:04 am
small changes on styles		Lucas-Qin	25/09/2022 11:03 am
footer logo fixed		Lucas-Qin	25/09/2022 10:49 am
Update README.md		Lucas Qin*	25/09/2022 10:42 am
navigation bar added		Lucas-Qin	25/09/2022 10:41 am
files restructured, all swipers fixed		Lucas-Qin	23/09/2022 1:43 pm
Merge pull request #1 from lucas-project/local-work-lucas	origin/natalie	Lucas Qin*	20/09/2022 12:07 pm
test		Lucas-Qin	20/09/2022 11:59 am
restructure files		Lucas-Qin	19/09/2022 5:36 pm
Merge branch 'main' of https://github.com/lucas-project/Innovation_project		Lucas-Qin	19/09/2022 5:18 pm
one swiper success		Lucas-Qin	19/09/2022 5:18 pm
Update README.md	origin/master	Lucas Qin*	19/09/2022 4:04 pm
Create README.md		Lucas Qin*	19/09/2022 3:52 pm
First version		Lucas Qin*	19/09/2022 3:50 pm

Fig.7.2 Technical log part-2

The github link for the project is
https://github.com/lucas-project/Innovation_project.git