



Lyffe

Project Engineering

Year 4

Aidan Shields

Bachelor of Engineering (Honours) in Software and
Electronic Engineering

Atlantic Technological University

2022/2023

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

_____Aidan Shields_____

Table of Contents

1	Summary	5
2	Poster	6
3	Acknowledgements.....	7
4	Introduction	8
5	Research.....	9
5.1	Requirements for a successful workout app.....	9
5.2	Code Stack	9
6	Project Architecture and Technologies.....	10
6.1	MongoDB.....	10
6.2	Express.js	11
6.3	React.js	11
6.4	Node.js.....	11
6.5	Material UI.....	11
6.6	Auth0.....	12
6.7	RapidAPI	12
7	Project Plan	14
7.1	Agile Methodology	14
7.2	Jira	14
7.3	Bitbucket	15
8	Development and Implementation	15
8.1	Frontend.....	15
8.2	Exercises Component.....	17
8.3	Workout Component	21

8.4	Third-Party Integrations	28
8.5	Backend	30
9	Ethics	37
9.1	Physical Health	37
9.2	Mental Health	37
10	Conclusion	38
11	References	38

1 Summary

The Lyffe project is a MERN (MongoDB, Express, React, Nodejs) full-stack web application designed to be a straightforward experience by providing exercise information and workout plans to the user. The goal is to offer an app that helps users of all levels work towards new goals.

The scope includes an appealing, friendly user interface that shows how to undertake an exercise, which muscle group the target group, along with progress tracking of workouts.

The project's features include the use of RapidAPI's ExerciseDB, which provides a database of exercises that can be done with or without equipment. The workout tracker allows the user to create a workout, keep track of their progress or change their plan by allowing to edit any of the fields desired.

The main technologies used in the project are the MERN stack which runs on JavaScript and Material UI's component library. React.js is the main framework used for the frontend, Express.js handles the requests and responses to the backend. MongoDB handles the data storage with Node.js being used as the runtime environment which executes the JavaScript code.

The project achieves a responsive design with the creation of a workout plan that caters to all levels, providing information on how to improve form with the aid of ExerciseDB from Justin Mozley on the RapidAPI platform.

In conclusion, Lyffe is a web application that provides information, and the creation of workout plans to help users achieve their fitness goals. The app accomplishes a responsive design and provides a comprehensive workout plan that caters to all users.

2 Poster

Description

Lyffe is a cutting-edge MERN full-stack web application designed to provide an all-inclusive workout plan to fitness enthusiasts. With its React-based Material UI library, it offers a comprehensive list of exercises that can be done with or without equipment, catering to all fitness levels and requirements.

Designed with a passion for fitness, Lyffe aims to help individuals overcome the daunting experience of starting at the gym by offering them a user-friendly platform to kickstart their fitness journey. From beginners to advanced goers, the app provides step-by-step guidance on how to undertake each exercise and which body parts they target.

By offering an alternative exercise program to gym-goers, Lyffe aims to inspire and motivate individuals to achieve their fitness goals. Its user-friendly interface and comprehensive workout plan make it an ideal choice for fitness enthusiasts looking to take their training to the next level.

Aidan Shields
BEng(H) Software
Electronic Engineering

G00370587 Lyffe

Technologies

Front-end: Dynamic web applications are constructed using the React framework for Javascript and Material UI, which provides access to a pre-made React component library for developing standardized and responsive UI components.

Back-end: Express is a popular framework used in web applications for handling HTTP requests and responses. Storage is done with MongoDB which is a noSql database. The server-side code is written in Javascript inside the Node.js environment

Node: Node.js is an open-source runtime environment for executing JavaScript code outside of a web browser. It uses an event-driven, non-blocking I/O model to allow for scalable and efficient server-side applications.

Api: RapidAPI is a marketplace that allows developers to discover, test, and connect to thousands of APIs. Auth0 is a cloud-based identity platform that provides secure authentication and authorization services, including single sign-on (SSO), multifactor authentication (MFA), and user management.

Architecture Diagram

```
graph LR
    Client[Client] <--> FE[Front End]
    Client <--> BE[Back End]
    FE <--> BE
    FE <--> S[Services]
    BE <--> S
```

Conclusion

The Lyffe web app is a web application designed for fitness enthusiasts. The front-end is built with React and Material UI, while the back-end uses Express and MongoDB for data storage. Node.js is used for server-side development, and RapidAPI and Auth0 are integrated for API management and secure authentication.

The app provides a comprehensive workout plan with step-by-step guidance on over 1400 exercises, catering to users of all fitness levels.

3 Acknowledgements

I'd like to thank all the lecturers for their help throughout the year, especially my mentor Michelle for her guidance and help throughout the year. I would also like to thank the rest of my class as a group, through the year everyone has been more than willing to help anyone in any area they can when able and has been a great class to be a part of.

Grammarly was used in the writing of this report, to check for typos and provide better wording in places.

4 Introduction

Health and fitness have always been important, but in recent years there has been a massive spike in people trying to improve both. As an avid gym goer, I wanted to build something that I would enjoy using myself. I began thinking about when starting in the gym, it was easy to find programs online but with no information on how to undertake the exercises. This inspired the decision behind Lyffe, to build something that would not only allow the user to plan out their week of workouts but find any exercise alternatives and get an understanding of how to do those exercises.

The scope was about creating a user friendly and visually appealing user interface. Involving a comprehensive list of exercises, along with being able to create, delete and update it with their progress. The app was built using the MERN stack and was integrated with RapidAPI's ExerciseDB for the comprehensive exercise database with the visuals required. Material UI was used for the design to create the user-friendly interface, Additionally Auth0 was used to implement Authentication.

This report will detail the entire process, starting with background on the importance of health and fitness, followed by an outline of technologies used the design and implementation process, followed by some testing and validation of the app. Finally, the report will look at the outcome and future developments.

5 Research

5.1 Requirements for a successful workout app

A user-friendly interface that leads to intuitive use, this being built on a good layout that makes it easy for users to navigate and provides the best experience for users of all levels. This meant researching what makes an app intuitive. This gave a better understanding of how to design the app. The app needs to allow users to easily find what they want. It needs to use the right icons to guide users, and finally it needs to allow them to predict the outcome of their action. This led to the decision to keep the app minimal and try to generate a flow that would allow users to improve their knowledge and move on to creating their workouts with this improved knowledge.[1]

5.1.1 Exercise Database

To be able to improve the user's knowledge of an exercise they may be unfamiliar with, a database containing that information is needed that would show how to do the exercise along with what equipment is required.

5.1.2 Customizable Workout Plan

This needed to cater to all levels, offering a good template to allow users to build their workout. It needed to offer full customisation to allow the user to not only create a workout but also control every aspect of the plan along with being able update the workout as they progress through towards their goals.

5.2 Code Stack

There are a few different aspects when deciding on a code stack. The project requirements, which in the Lyffe project the needs were a visually appealing UI, database integration, support and finally the competence of the developers. Having previous experience with React, I decided to look further into it more and research the MERN and MEAN stacks. [2]

5.2.1 MEAN(MongoDB, Express.js, Angular, and Node.js)

The MEAN stack is a popular choice for web applications that offers, scalability and performance. While it is powerful and robust, due to the time it would take to learn angular it didn't make sense to use this stack.

5.2.2 MERN (MongoDB, Express.js, React.js, Node.js)

The MERN stack met all the necessary requirements, a big benefit was previous experience with React which lessened the learning curve. It runs solely on the JavaScript language. It allowed for database integration and had extensive community support that provides updates and support future development. With further research at UI, I found that component libraries existed, which solidified the use of the MERN stack as it allows for a consistent component style throughout the app.

Further research done into the project has been expanded into sections their own sections, where a more detailed look at the use of them in the project can be found.

6 Project Architecture and Technologies

Lyffe uses a Single Page Application (SPA) architecture with React.js as the main front end framework. SPA architecture provides a better user experience by dynamically updating the contents without fully reloading the page.

Lyffe consists of these main components:

Auth0: used for the authentication and route authorisation,

App.js and index.js: Create the entry points of Lyffe and set out the structure and routing.

Exercises.js: Manages the search and display along with the detail display functionality.

Workout.js: Contains the creation and modification of the workout plans.

6.1 MongoDB

MongoDB is a document database used to build universally available and scalable internet applications. It is a NoSQL database in a JSON format called BSON.[3] In Lyffe, MongoDB is used

for the data storage of workouts for each user. It stores the data in a document format. It makes it easy for to search and retrieve the workout data through the userId.

6.2 Express.js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.[4] Lyffe uses Express to manage the HTTP requests and responses for the API endpoints related to work out plans.

6.3 React.js

React is a JavaScript library for rendering user interfaces (UI). It promotes the use of components which makes for a modular codebase[5]. React.js is used in Lyffe to create the front end. It allows for the responsive and user-friendly design that was set out.

6.4 Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project.[6] In Lyffe Node.js is used as the runtime environment which executes the JavaScript code that handle's the HTTP requests.

6.5 Material UI

Material UI is a React UI framework that follows Google's Material Design guidelines. It gives the developer prebuilt components that are ready to use once imported[7] Material UI components were used throughout the front end to give the app a consistent style for a better user experience.

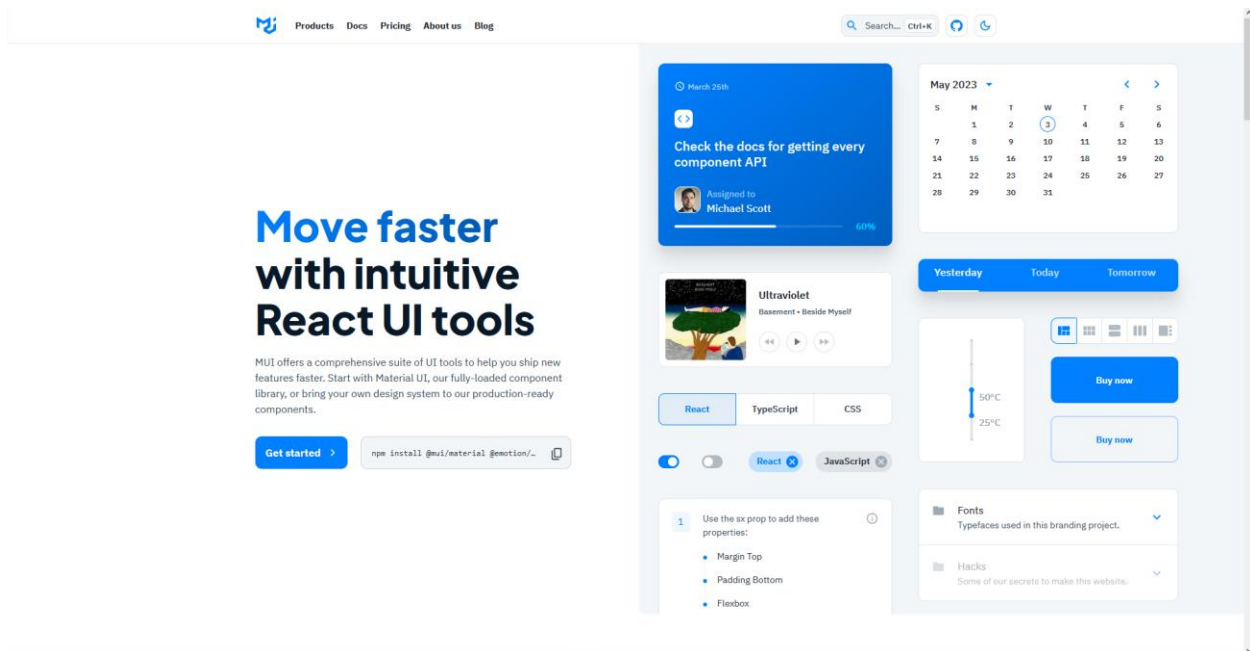


Figure 5-1 Material UI

6.6 Auth0

Auth0 is a drop-in solution to add authentication and authorization services to applications. [8] Auth0 provides user authentication and authorization in Lyffe.

6.7 RapidAPI

RapidAPI is a popular API marketplace that provides a platform for developers to discover, connect to, and manage APIs in their projects.[9] In Lyffe, ExerciseDB from Justin Mozley, provides the Comprehensive database of exercises containing the visual representation described in the scope.[10]

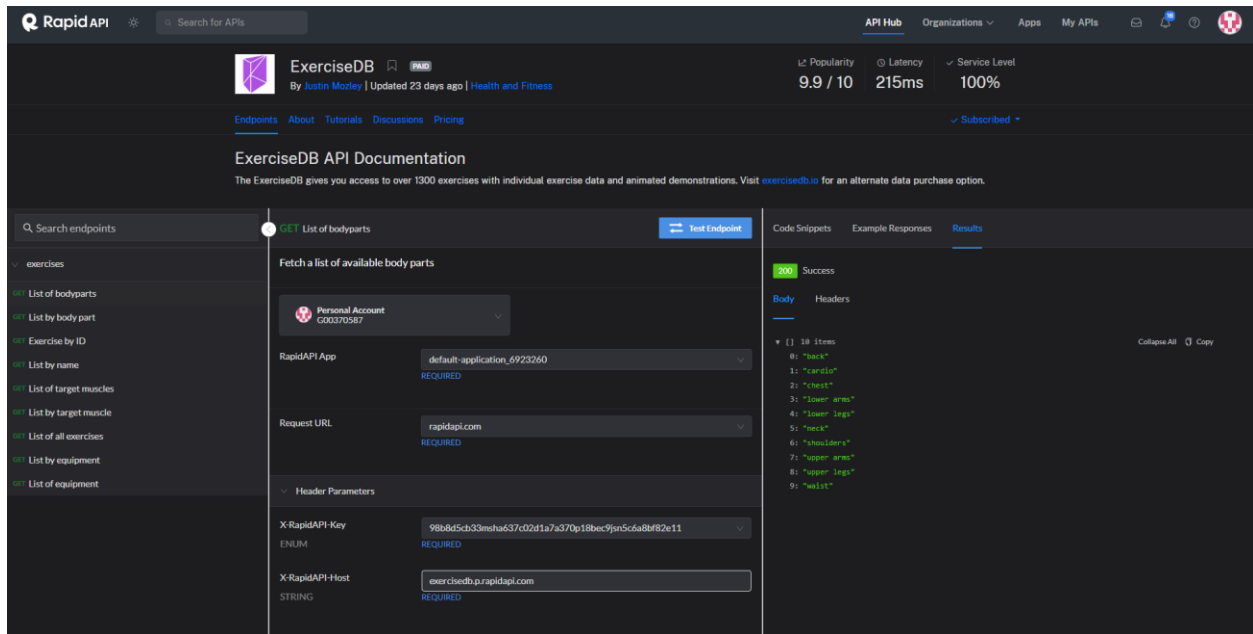


Figure 6-2 ExerciseDB on RapidAPI platform

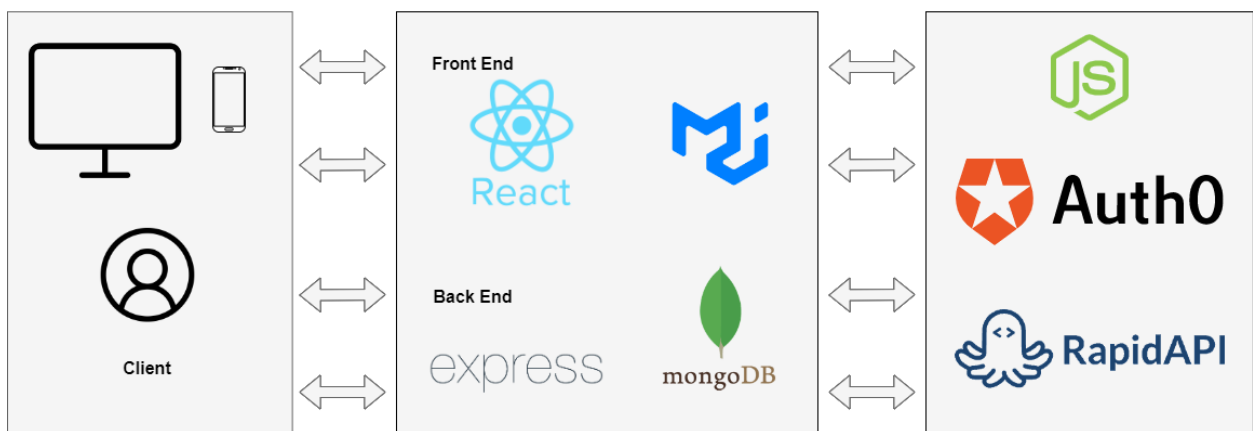


Figure 6-2 Architecture Diagram

7 Project Plan

For the project planning the Atlassian stack was used, Jira and Bitbucket [11]. Jira is a project management tool that's used for issue tracking and agile project management. Bitbucket is a web-based version control repository service to manage source repository.

7.1 Agile Methodology

Kanban is the agile methodology that was used for the project development, it focuses on flexibility and delivery. Workflow in kanban is about efficiency and is done by limiting the number of tasks that are in progress at any one time. This promotes the developer to stay focused on the open tasks, in turn this means an epic will move towards completion in a more efficient method.

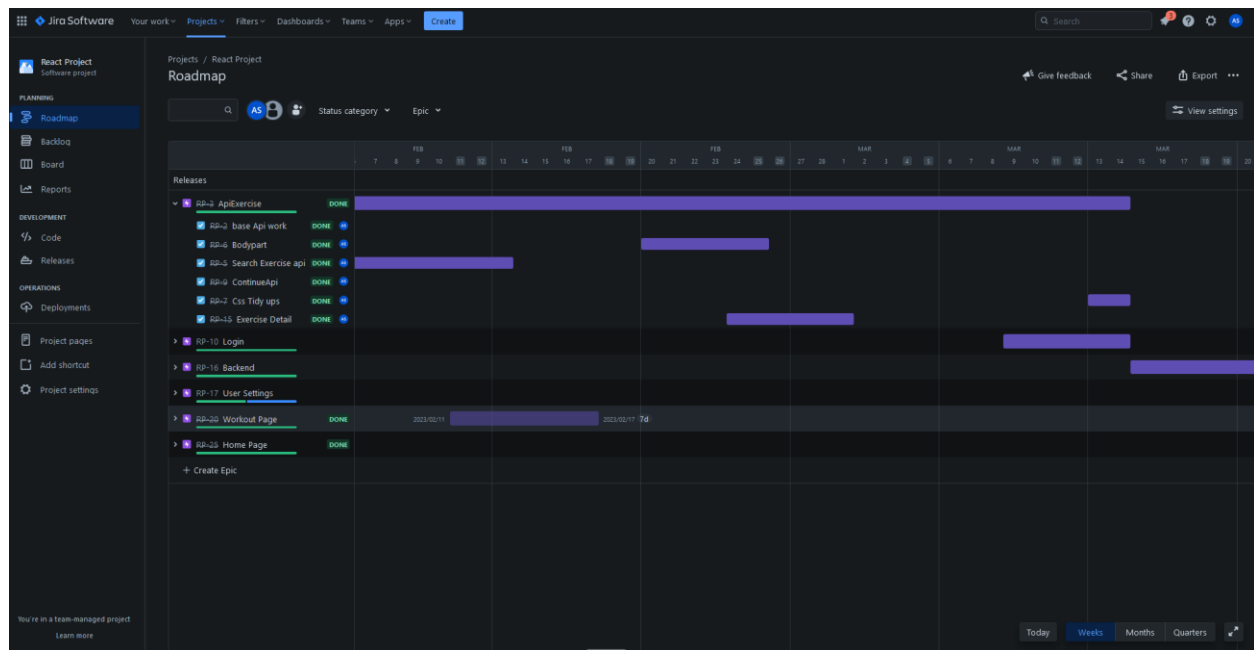


Figure 7.1.1 – Jira Roadmap

7.2 Jira

Jira was used for workflow and issue tracking with the Kanban method being the agile methodology. Using a Kanban board, the project was broken down into epics and created tasks within those epics, priorities were set in those tasks, with the developer deeming the tasks a

priority from highest to lowest. From here that promotes the kanban method by creating a natural flow of tasks from start to finish of an epic. Within the board, four stages were created, “To Do”, “In Progress”, “In Review” and “Done”. Each stage represents a task in different stages of development. Automation tools were used to automatically trigger events within the board, when a branch was created the corresponding card moved from “To Do” to “In Progress” then moved on to “In Review” when the pull request was created then on completion of the merge request the card moved to “Done” and would be marked as done in the epic. Following on with the Kanban, only a maximum three tasks were ever allowed to be in progress by design to ensure work was completed in a timely manner. [12]

7.3 Bitbucket

Bitbucket is where the Lyffe repository is stored. It is a web-based hosting service that provides version control and allows collaboration for development teams. It uses the branching method that is popular today. This allowed for each task to be developed and reviewed before being merged back into the main branch, Bitbucket provides information on any code conflicts it finds also, this combined with the branch method hugely decreases the chances of the main branch ever breaking.

Overall, the use of the Atlassian Stack allowed for seamless workflow and an easy-to-follow plan for development process by combining the management and the repository. [13]

8 Development and Implementation

This section will look at the design and implementation of Lyffe in detail. It will look at the important elements of the project in both the front end, backend, and third-party integrations.

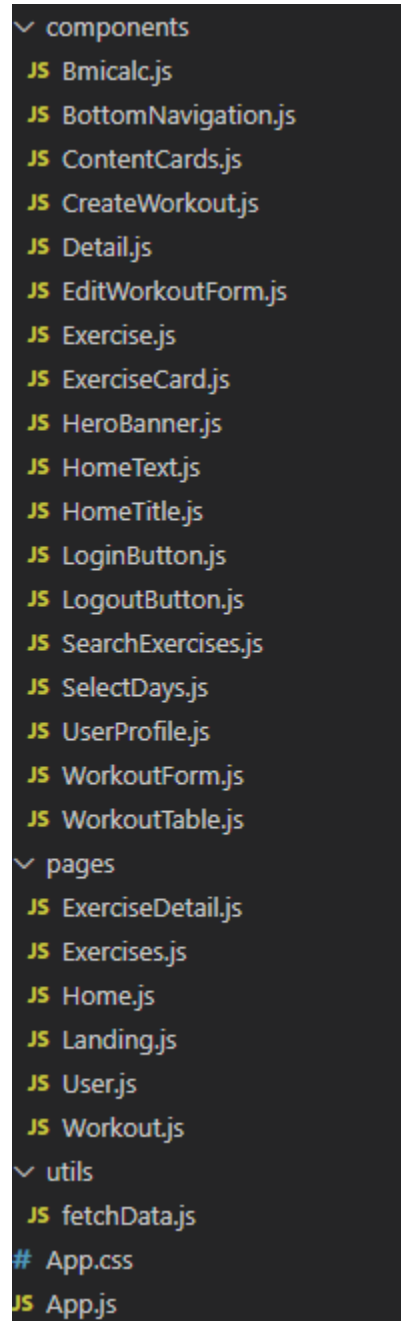
8.1 Frontend

The frontend of Lyffe is developed using React.js. Its component architecture allows it to be broken down into modular components, which improves maintainability and scalability.

8.1.1 Frontend Structure

The front end is split into three main sections. The App.js which is where the core of the app is contained, the screens which contain the components and then the components themselves.

The components are independent and are reusable throughout the app allowing for scalability.



```
✓ components
  JS Bmicalc.js
  JS BottomNavigation.js
  JS ContentCards.js
  JS CreateWorkout.js
  JS Detail.js
  JS EditWorkoutForm.js
  JS Exercise.js
  JS ExerciseCard.js
  JS HeroBanner.js
  JS HomeText.js
  JS HomeTitle.js
  JS LoginButton.js
  JS LogoutButton.js
  JS SearchExercises.js
  JS SelectDays.js
  JS UserProfile.js
  JS WorkoutForm.js
  JS WorkoutTable.js
✓ pages
  JS ExerciseDetail.js
  JS Exercises.js
  JS Home.js
  JS Landing.js
  JS User.js
  JS Workout.js
✓ utils
  JS fetchData.js
# App.css
JS App.js
```

Figure 8.1.1 – Frontend Structure

Inside App.js the routes are defined for each screen using React-Router-Dom. This enables the creation of routes by defining and mapping them to the specific components they are assigned to. In Lyffe, it is used for five main routes, which are shown in the pages in figure 8.1.2. It also allows for the transition between different views without full page reloads. The routing is configured using the <Routes> and <Route> tags. In App.js there are two different routes components which protects some routes in a ternary operator so that they can only be accessible when the user is authenticated.

```

    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/exercises" element={<Exercises />} />
      <Route path="/workout" element={<Workout />} />
      <Route path="/user" element={<User />} />
      <Route path="/exercise/:id" element={<ExerciseDetail />} />
    </Routes>

  </Box>
  <BottomNavigation />
</div>
) : (
  <Routes>
    <Route path="/" element={<LandingPage />} />
  </Routes>
)}

```

Figure 8.1.2 – Routes Structure

8.2 Exercises Component

The Exercise and SearchExercises components make up the Exercises page, this page enables the user to discover and search exercises by multiple aspects.

8.2.1 Exercise Component

The exercise component displays the list of exercises and manages pagination.

To get the exercise data from RapidAPI the fetchExercise function is called to fetch data from

the API endpoint that is on the Data object from the utils/fetchData.js. The endpoint and key are sent in the headers. It then stores the returned data in the exercises state, with the key being stored in an environment variables file. This all is in a useEffect that that has an empty dependency which means it run one the component loads.

```
useEffect(() => {  
  const fetchApiExera = async () => {  
    let exData = [];  
    exData = await fetchExercise('https://exercisedb.p.rapidapi.com/exercises', Data);  
    setData(exData);  
  };  
  
  fetchApiExera();  
}, []);
```

Figure 8.2.2 – FetchAPI within the useEffect

This component is then rendered into a 'Box' Material UI component, this uses a flexbox model for arranging elements. The “Stack” component also is a layout component that allows for items to be stacked vertically or horizontally, as well as allowing for spacing Here, the ExerciseCard component is rendered evenly spaced in a row from left to right. This then maps through the currentExercises array, the map() method iterates over each item in the array. For each item it iterates over it returns a new ExerciseCard component with two props “key” and “exercise”. This is how the cards be rendered in the same component with different information.

```

<Box id="ex" className="exercise-box" sx={{ mt: { lg: '109px' } }}>
  <Stack className="exercise-stack" direction="row" sx={{ gap: { lg: '107px', xs: '50px' } }}>
    {currentExercises.map((exercise, idx) => (
      <ExerciseCard key={idx} exercise={exercise} />
    ))}
  </Stack>
  <Stack sx={{ mt: '70px' }} alignItems="center">
    {ex.length > 9 && (
      <Pagination
        color="standard"
        shape="rounded"
        defaultPage={1}
        size="medium"
        page={currentPage}
        onChange={paginate}
        count={Math.ceil(ex.length / exercisesPerPage)}
      />
    )}
  </Stack>
</Box>

```

Figure 8.2.3 – Render of ExerciseComponent.

8.2.2 SearchExercises

The SearchExercises component allows the user to search through the exercises based on their name, body part, muscle, or equipment.

Here the users can search through a “TextField” component, the “setSearch” function then updates its state with the users input and converts it to lowercase. The “onChange” event listener on the input field triggers the “setSearch” to update the state with the text input and then calls the “handleSearch” function onclick. “handleSearch” is the function that filters the exercises based on the search input. First it checks if the query exists, it then calls the “fetchExercise” endpoint. The searched array gets filled by filtering the data, then checks the query against the name, body part, muscle, or equipment.

```
const handleSearch = async () => {
  if (search) {
    const exData = await fetchExercise('https://exercisedb.p.rapidapi.com/exercises', Data);

    const searched = exData.filter(
      (data) => data.name.toLowerCase().includes(search)
      || data.target.toLowerCase().includes(search)
      || data.equipment.toLowerCase().includes(search)
      || data.bodyPart.toLowerCase().includes(search),
    );

    setSearch('');
    setData(searched);
  }
};
```

Figure 8.2.4 – handleSearch Function

In summary the Exercise and SearchExercise components provide an efficient way for users to search and discover new exercises. The two components interact with RapidAPI through the fetchExercise endpoint which returns data and updates the state of the application.

```
<Stack className="search-stack">
  <Typography fontWeight={700} sx={{ fontSize: { lg: '44px', xs: '30px' } }} mb="49px" textAlign="center">
    Search for any Exercise or Bodypart!
  </Typography>
  <Box className="search-box">
    <TextField
      className="search-text"
      sx={{ input: { fontWeight: '700', border: 'none', borderRadius: '4px' }, width: { lg: '1170px', xs: '350px' }}}
      value={search}
      onChange={(e) => setSearch(e.target.value.toLowerCase())}
      placeholder="Search"
      type="text"
    />
    <Button className="search-btn" sx={{ textTransform: 'none', width: { lg: '173px', xs: '80px' }, fontSize: { lg: '20px', xs: '14px' } }} onClick={handleSearch}>
      Search
    </Button>
  </Box>
</Stack>
```

Figure 8.2.5 –Search exercise “onChange”

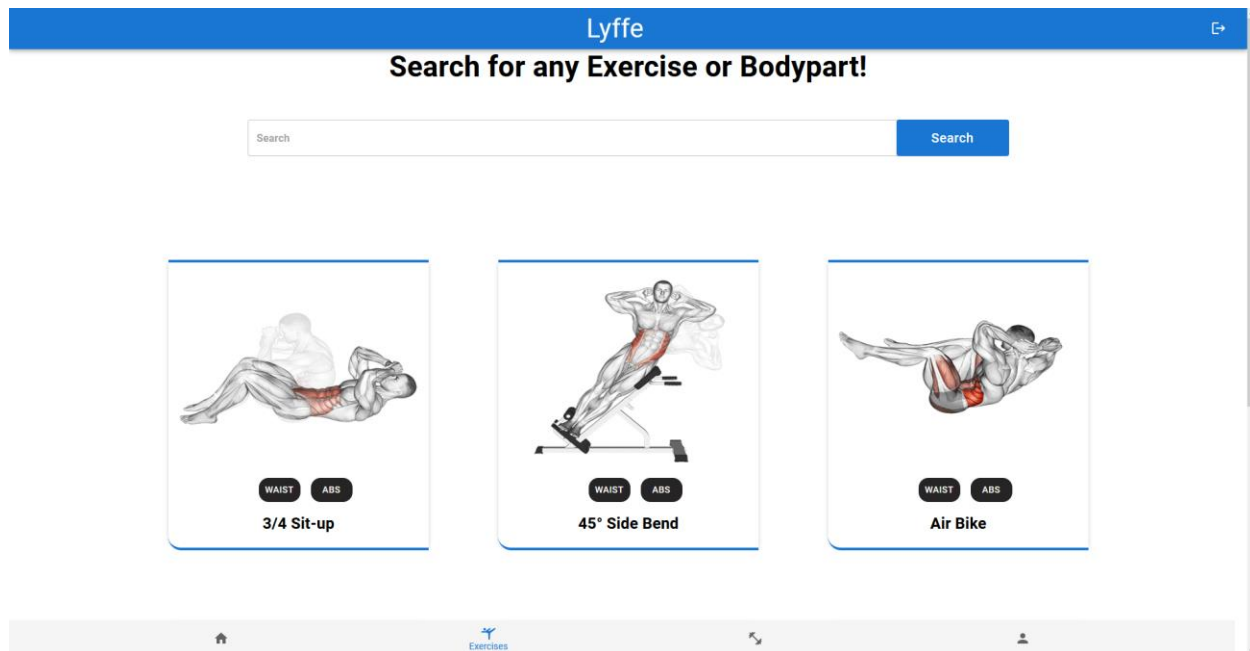


Figure 10.1.1 – Exercises

8.3 Workout Component

The Workout Component handles the user's workouts by allowing the user to create, update, and delete workout plans with their specific userId. In the Workout component, there are three main components: 'CreateWorkout', 'WorkoutForm' and 'WorkoutTable'. On load of the Workout page, a useEffect hook runs an async 'fetchData.' If the user object exists, this function calls the 'fetchUserWorkout' with the user's Id. The 'handleWorkoutCreated' is a callback function for when a successful workout is created. This function runs and passes back the value which sets the 'showCreateWorkout' hook to false. It Then fetches the user's workout. By doing this, the 'userWorkout' object is created, and the 'WorkoutTable' renders.

```

const { user, getAccessTokenSilently } = useAuth0();
const [userWorkout, setUserWorkout] = useState(null);
const [showCreateWorkout, setShowCreateWorkout] = useState(false);

const fetchData = async () => {
  const fetchedWorkout = await fetchUserWorkout(user.sub);
  setUserWorkout(fetchedWorkout);
};

useEffect(() => {
  if (user?.sub) {
    fetchData();
  }
}, [user]);

const handleWorkoutCreated = () => {
  setShowCreateWorkout(false);
  fetchData();
};

```

Figure 8.3.1 – Workout states and useEffectHooks.

This is where the conditional rendering comes into the page, if there is no workout it will display text and a button to create the workout, onclick for that button it sets the “CreateWorkout” component to visible and the user can then begin to create their workout. If there is a workout it will pass the workout data to the “WorkoutTable” component.

```

{!userWorkout && !showCreateWorkout && (
  // Render the message and button to create a workout if there's no workout
  <div className="no-workout">
    <Typography variant="h3" gutterBottom>
      No Workout Data
    </Typography>
    <Button variant="contained" onClick={() => setShowCreateWorkout(true)}>
      Create Workout
    </Button>
  </div>
)}
{userWorkout && (
  // Render the WorkoutTable component if there's a workout
  <div>
    <WorkoutTable workoutData={userWorkout.workouts} userId={user.sub} onDelete={handleDelete} onWorkoutUpdated={fetchData} />
  </div>
)}
{showCreateWorkout && (
  // Render the CreateWorkout component to create a new workout
  <CreateWorkout onWorkoutCreated={handleWorkoutCreated} />
)}

```

Figure 8.3.2 – conditional rendering

8.3.1 CreateWorkout Component

Users select the number of days to work out in the “SelectDays” component, “workoutDays” gets passed from here to “CreateWorkout”, then an array is created to the length of the number in “workoutDays”. From here a new component “WorkoutForm” created for each index in the Array.

```
{!workoutDays && <SelectDays onDaysSelected={onDaysSelected} />}
{workoutDays &&
  Array.from({ length: workoutDays }, (_, idx) => (
    <Accordion key={idx} className="workout-accordion">
      <AccordionSummary expandIcon={<ExpandMoreIcon />} className="workout-title">
        Workout {idx + 1}
      </AccordionSummary>
      <AccordionDetails className="workout-accordion-details">
        <WorkoutForm key={idx}
          onFormChange={({formData}) => handleWorkoutFormChange(formData, idx)} />
      </AccordionDetails>
    </Accordion>
  ))}
{workoutDays && (
  <Button variant="contained" color="primary" onClick={handleFormSubmit}>
    Save Workouts
  </Button>
)}
```

Figure 8.3.3 – CreateWorkout conditional rendering

8.3.2 WorkoutForm

This component allows the user to enter all the necessary fields to create the proper workout plan.

To do that an array of objects was created called “formData”. This also takes the “numRows” which is set at a base to 6.

```
const [formData, setFormData] = useState(
  Array(numRows)
    .fill()
    .map(() => ({
      exercise: "",
      sets: "",
      reps: "",
      weight: "",
    })))
);
```

Figure 8.3.4 – formData Array.

To update the state of the “formData”, “handleInputChange” was created. It takes three parameters “event, rowIndex, field”. Event is the event object, “rowIndex” is the index for the row in “formData” and field is the fields that is updated on input.

```
const handleInputChange = (event, rowIndex, field) => {
  const newFormData = [...formData];
  newFormData[rowIndex][field] = event.target.value;
  setFormData(newFormData);
  onFormChange(newFormData);
};
```

Figure 8.3.5 – handleInputChange.

```
{formData.map((row, rowIndex) => (
  <div key={rowIndex} className="workout-row">
    <Grid container spacing={2} className="workout-form-container">
      <Grid item xs={12} md={3}>
        <TextField
          label="Exercise Name"
          variant="outlined"
          value={formData[rowIndex]?.exercise || ""}
          onChange={(event) =>
            handleInputChange(event, rowIndex, "exercise")
          }
        />
      </Grid>
    </Grid>
  )
)}
```

Figure 8.3.6 – Textfield for exercise

8.3.3 WorkoutTable Component

This component is responsible for displaying the user’s workout data. It contains the “EditWorkoutForm” component that loads in the selected workout using its “workoutId” and allows the user to update individual fields in the specific workout, it also manages the “handleEdit”, “handleSave”, and “handleCancel” functions which are for editing specific workouts, saving those changes and deleting the workout.




Lyffe			
Workout 1 			
Exercise Name	Sets	Reps	Weight (kg)
Barbell Back Squat	3	10	90 kg
Barbell Bench Press	4	8	85 kg
Pull ups	4	8	0 kg
Dumbbell Lateral Raises	4	10	10 kg
Nordic Hamstring Curls	3	10	40 kg
Calve Raises	4	10	50 kg
Workout 2 			
Exercise Name	Sets	Reps	Weight (kg)
Dumbbell Shoulder Press	4	8	27.5 kg
Deadlift	4	5	100 kg
Incline Dumbbell Press	3	12	25 kg
Machine Cable Rows	3	12	60 kg
Weighted Tricep Dips	3	12	10 kg
Bicep EZ bar Curls	3	13	50 kg
Workout 3 			
Exercise Name	Sets	Reps	Weight (kg)
Machine Knee Extension	3	10	45 kg
Bulgarian Split Squats	3	10	15 kg

Figure 10.1.2 – Workout Component

```

const handleSave = async (newData) => {
  try {
    const accessToken = await getAccessTokenSilently();
    const success = await saveUserWorkout(
      userId,
      editingWorkout.workoutId,
      newData,
      accessToken
    );

    if (success) {
      onWorkoutUpdated();
      setEditingWorkout(null);
    }
  } catch (error) {
    console.error("Error updating workout:", error);
  }
};

const handleCancel = () => {
  setEditingWorkout(null);
};

const [showDeleteButton, setShowDeleteButton] = useState(true);

useEffect(() => {
  setShowDeleteButton(editingWorkout === null);
}, [editingWorkout]);

return (
  <div className="table-root">
    {editingWorkout ? (
      <EditWorkoutForm
        workoutData={editingWorkout.ex}
        onSave={handleSave}
        onCancel={handleCancel}
        onWorkoutUpdated={onWorkoutUpdated}
      />
    ) : (

```

Figure 8.3.7 – EditWorkoutForm and handles

“workoutData” is passed to the WorkoutTable component as an object then using “Object.entries(workoutData).map()” function it iterates over each object to create a new table for each, within that table it uses “Object.values(ex).map()” which iterates over the “ex” which are the exercises and populates each row according to its matching “rowIndex”.

```

    Object.entries(workoutData).map(
      ([workoutId, ex], workoutIdx) => (
        <div key={workoutIdx} className="table-row">
          <Typography variant="h5" className="workout-heading">
            Workout {workoutIdx + 1}
          </Typography>
          <IconButton ...
          </IconButton>
          <Grid container spacing={2} className="table-form-con">
            <Grid ...
            </Grid>
            <Grid
              item ...
            >
              <Typography variant="subtitle1">Sets</Typography>
            </Grid>
            <Grid ...
            </Grid>
            <Grid ...
            </Grid>
            {Object.values(ex).map((exercise, rowIndex) => {
              if (typeof exercise === "object") {
                return (
                  <React.Fragment key={rowIndex}>
                    <Grid
                      item
                      xs={12}
                      md={3}
                      className={`table-grid ${
                        rowIndex % 2 === 0 ? "table-grid-row" : ""
                      }`}
                    >
                      <Typography>{exercise.exercise}</Typography>
                    </Grid>

```

Figure 8.3.8 - Render of WorkoutTable

The Workout component and its subcomponents provide the user with the ability to create and modify workout plans within a friendly interface.

8.4 Third-Party Integrations

This section details how the integration of third-party API's can be done and how they are used in the Lyffe project.

8.4.1 Auth0

Lyffe uses Auth0 for authentication across the app. It provides a secure solution for handling profile management including registration, login, and logout. For Auth0 to be integrated into Lyffe, first the developer must first sign up to Auth0. Then follow the steps to create an application in the dashboard, configure the application with the correct settings for the application. For Lyffe it was a template for a React.js SPA.

To integrate Auth0 into the application, the auth0-react package is used. The Auth0Provider component wraps around the root component in Index.js so that Auth0 functionality can be used throughout the application.

The “getAccessTokenSilently” method, provided by auth0 through the useAuth0 hook, is main method for securing routes with auth0. The method retrieves the access token from the authenticated user, which is passed as a bearer token in the header. This token then gets verified in the backend by the “verifyAccessToken” method. [14]

```

import { BrowserRouter } from 'react-router-dom';
import { Auth0Provider } from '@auth0/auth0-react';

const domain = process.env.REACT_APP_AUTH0_DOMAIN;
const clientId = process.env.REACT_APP_AUTH0_CLIENT_ID;

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <Auth0Provider
      domain = {domain}
      clientId = {clientId}
      authorizationParams={{
        redirect_uri: window.location.origin
      }}
    >
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Auth0Provider>
  </React.StrictMode>
);

```

Figure 8.4.1 – Auth0 Provider

8.4.2 ExerciseDB / RapidAPI

Lyffe uses ExerciseDB which is a RESTful API for the database of exercises containing the visual representation that would provide the visual information about the exercises. It provides one of the main features of the app through the detailed information on the exercises. It gives the exercise names, their target muscle, which body part it is working and whether any equipment is needed.

It is called using the `fetchExercise` utility function that handles the API request to the API database. The function takes in two parameters, the API endpoint and the object `Data` which contains the headers and key. It uses the `fetch` API function to make the requests and gets a JSON response that contains the data.

```
export const Data = {  
  method: 'GET',  
  headers: {  
    'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com',  
    'X-RapidAPI-Key': process.env.REACT_APP_RAPID_API_KEY,  
  },  
};  
  
export const fetchExercise = async (url, options) => {  
  const res = await fetch(url, options);  
  const data = await res.json();  
  
  return data;  
};
```

Figure 8.4.2 – Utility Function and Data object

8.5 Backend

The backend of Lyffe is developed using Node.js, Express.js and uses MongoDB. These provide efficient server-side technologies that are essential for the workout plans and user authentication.

8.5.1 MongoDB Models and Schemas.

There are two schemas used in Lyffe's backend, workoutSchema and workoutPlanSchema. Using Mongoose, the shape of the schemas can be defined, each workout schema is required to have an exercise, reps, sets, and weight. The workoutPlanSchema defines the structure of a workout plan. It requires the userId, the number of days and the objects of the workouts. There is validation to check to make sure the days set falls between the set values that match the front-end choice.

```
const workoutSchema = new mongoose.Schema({
  exercise: { type: String, required: true },
  reps: { type: Number, required: true },
  sets: { type: Number, required: true },
  weight: { type: Number, required: true },
});

const workoutPlanSchema = new mongoose.Schema({
  userId: { type: String, required: true },
  days: {
    type: Number,
    required: true,
    validate: {
      validator: function (value) {
        return value >= 3 && value <= 6;
      },
      message: 'Days must be a number between 3 and 6',
    },
  },
  workouts: { type: Map, of: [workoutSchema] },
});
```

Figure 8.5.1 – Lyffe Schemas

The last line in the workoutPlanSchema took a long time to get the workouts saving correctly. The workouts property is of type “Map” which is a collection of key-value pairs and each key maps to a specific value. In here each value gets mapped to the correct. Here the keys are the “workoutIds” created in the /addworkout route. “of” specifies the type of value stored in the map. Here the values are array of the workoutSchema objects that have been sent from the front end. This is what allows for the add, update or removal of workouts from the plan while each value is validated against the workoutSchema.[15], [16]

8.5.2 Authorisation

In Lyffe’s backend Auth0 is used again for the authorisation of the user to access the routes. From research into the Auth0 documentation it showed how to verify the access token using the user info from the respective database. It’s used in each of the routes to validate if the user is authorised to make the request. In the routes/workouts.js file there is an asynchronous function called “verifyAccesstoken”. This function sends a get request to the Auth0

management API “/userinfo”. [17] The request sends the access token in the authorisation header, formatted as a bearer token. A bearer token is a keyword that indicates the type of authentication being used. If successful, the response returns the users sub which is a unique identifier for each user. If invalid, the catch allows the code to keep running, .

```
const verifyAccessToken = async (accessToken) => {
  try {
    const response = await axios.get(`https://${process.env.AUTH0_DOMAIN}/userinfo`, {
      headers: { Authorization: `Bearer ${accessToken}` },
    });

    const { sub } = response.data;
    return sub;
  } catch (error) {
    console.error('Error verifying access token:', error);
    return null;
  }
};
```

Figure 8.5.2 – User Authorization

8.5.3 /addworkout Route

This route is a post route responsible for the creation of the workout. In each route the first thing done is the access token is retrieved from the header. The header is split to remove the “bearer” from the Incoming string and just use the token. then the token is verified through the verifyAccess token method, if successful the request containing the workout plan is then received as an object, then the map() function creates a new array using the uuid4() function for each workout. From here a new workoutplan is created containing the userId, the number of days they workout and the workouts, which now contains the new uuid’s with their workout objects. If successful, the status 200 is returned if there is an error the status 500 will be returned.


```

router.post('/addworkout', async (req, res) => {
  try {
    const authHeader = req.headers.authorization;
    const accessToken = authHeader.split(' ')[1];

    const sub = await verifyAccessToken(accessToken);

    if (!sub) {
      res.status(401).send('Unauthorized');
      return;
    }

    const workoutPlanData = req.body;

    console.log('Received workout data:', workoutPlanData);

    const workoutsMap = new Map(
      workoutPlanData.workouts.map((workout) => [uuidv4(), workout])
    );

    const workoutPlan = new WorkoutPlan({
      userId: sub,
      days: workoutPlanData.days,
      workouts: workoutsMap,
    });
    await workoutPlan.save();

    res.status(200).send('Workout plan saved successfully');
    console.log('Workout plan saved successfully');
  } catch (error) {
    console.error(error);
    res.status(500).send('Server error');
  }
});

```

Figure 8.5.3 – /addworkout route

8.5.4 /userId route

This is a get route responsible for fetching the user's workout. Again, as in the /addworkout route the token is verified, if successful Lyffe will search the database for a workout plan with

the users Id, if it exists it will send the workout plan to the front end to be displayed, otherwise a 404 not found error is sent.

```
router.get('/:userId', async (req, res) => {
  try {
    const userId = req.params.userId;

    const workoutPlan = await WorkoutPlan.findOne({ userId });

    if (!workoutPlan) {
      res.status(404).send('Workout plan not found');
      return;
    }

    res.status(200).json(workoutPlan);
  } catch (error) {
    console.error('Error fetching user workout:', error);
    res.status(500).send('Server error');
  }
});
```

Figure 8.5.4 – /userId route

8.5.5 /userId/workoutId route

This is a put route responsible for updating specific workout data in a user's workout plan. If authorized successfully, the `userId` and the `"workoutId"` are extracted from the parameters, and the workout data from the body. It then performs a MongoDB search using Express to retrieve a workout plan with that `workoutId`. Two checks are ran, one for the workout itself and one for the workout with that id. if successful, this retrieves a workout with the specified id from the parameters. It uses the `". get"` to retrieve the `"workoutId"` within its `"workoutplan"` then maps over the exercises and their index. For each exercise, the ternary operator checks for an updated exercise at the same index in the new workout data in the request. If there is, it returns the new data otherwise it returns the existing data. This creates a new array called `"updatedWorkout."` This means it only updates the fields that have been updated in the database.

```

router.put('/:userId/:workoutId', async (req, res) => {
  try {
    const authHeader = req.headers.authorization;
    const accessToken = authHeader.split(' ')[1];

    const sub = await verifyAccessToken(accessToken);

    if (!sub) {
      res.status(401).send('Unauthorized');
      return;
    }

    const { userId, workoutId } = req.params;
    const workoutData = req.body;

    console.log(`Updating workout ${workoutId} for user ${userId} with data:`, workoutData);

    const workoutPlan = await WorkoutPlan.findOne({ userId });

    if (!workoutPlan) {
      res.status(404).send('Workout plan not found');
      return;
    }

    if (!workoutPlan.workouts.has(workoutId)) {
      res.status(404).send('Workout not found');
      return;
    }

    const updatedWorkout = workoutPlan.workouts.get(workoutId).map((exercise, index) => {
      return workoutData[index] ? workoutData[index] : exercise;
    });

    workoutPlan.workouts.set(workoutId, updatedWorkout);
    await workoutPlan.save();

    res.status(200).send('Workout updated successfully');
    console.log('Workout updated successfully');
  } catch (error) {
    console.error(error);
    res.status(500).send('Server error');
  }
});

```

Figure 8.5.5 – /userId/:workoutId route

8.5.6 /userId route

This is a delete route that is responsible for deleting the workout for the user. Firstly, the checks are done for the access token, if successful, the userId is extracted from the parameters and it then uses the `.findOneAndDelete` query and deletes the workout plan from the database.

```
router.delete('/:userId', async (req, res) => {
  try {
    const authHeader = req.headers.authorization;
    const accessToken = authHeader.split(' ')[1];

    const sub = await verifyAccessToken(accessToken);

    if (!sub) {
      res.status(401).send('Unauthorized');
      return;
    }

    const userId = req.params.userId;

    const workoutPlan = await WorkoutPlan.findOneAndDelete({ userId });

    if (!workoutPlan) {
      res.status(404).send('Workout plan not found');
      return;
    }

    res.status(200).send('Workout plan deleted successfully');
    console.log('Workout plan deleted successfully');
  } catch (error) {
    console.error('Error deleting workout plan:', error);
    res.status(500).send('Server error');
  }
});
```

Figure 8.5.6 – /userId route

Collectively these routes enable Lyffe to manage the workout plans, allowing an authenticated user to work towards their goals.

9 Ethics

Exercising has been a long-standing pillar in creating a healthy lifestyle both physically and mentally. An active lifestyle has been proven to be crucial for health as we age. Regular exercise has been proven to reduce the risk of chronic disease and provide a high quality of life, here some of those topics will be detailed.

9.1 Physical Health

As we age our bodies lose mass. This is called Sarcopenia. It is a condition that is age-related progressive loss of muscle mass and strength[18]. Studies have shown that working out can help to hugely slow that down[19]. This in turn will provide a better quality of life as someone gets older. This is what Lyffe aims to promote, users aren't expected to lift very heavy weights but to aim at, improving the user's knowledge, so that it provides the best quality of "Lyffe" the user can experience. Of course, working out alone doesn't solve health all issues but it does aid in prevention and recovery for a huge amount.

9.2 Mental Health

Exercise has huge benefits for mental health that personally I feel gets overlooked. Studies have shown it reduces cognitive decline as a person begins to age[20]. For people who regularly workout, when struggling with their mental health, exercising can clear the mind and give a reset for the brain. Exercising can also provide a great place for those struggling to make friends, team sports provide opportunities to meet new people, along with the benefits from exercising it fosters a way to combat loneliness which is one of the major keys in the mental health area.

Overall, the benefits of exercising can't be overlooked it. Humans are the highest endurance runners on the planet, meaning they are designed to be moving, people aren't expected to run marathons but regular walks, runs or working out prove to be invaluable to a healthy body and mind and Lyffe is a tool that can hopefully help people on that journey.

10 Conclusion

In conclusion, Lyffe was developed successfully, combining backend functionality with a consistent, user-friendly interface that provides users with information and a comprehensive workout plan structure. The integration of third-party API's from Auth0 and RapidAPI allowed for a quicker process as time was of the essence in the the development of Lyffe.

The front end provides a clean, user-friendly interface to help users decide their workout plans and build towards their goals, Material UI aided massively in the design with the pre-built components that can be navigated using React Router Dom.

The backend which uses Express.js and MongoDB, takes care of the storage and retrieval of the workout data. All while the interactions of the front and backend being exceptionally smooth.

Lyffe is a platform that can help provide information and education to users. Along with both physical and mental benefits to its users of all ages.

11 References

- [1] 'How to Build a Fitness App: Step by Step Guide - CodeIT'. <https://codeit.us/blog/how-to-build-a-fitness-app#choose-the-fitness-app-type-you-need> (accessed Apr. 27, 2023).
- [2] 'MEAN vs MERN - The Ultimate Comparison 2023 | Clean Commit'. <https://cleancommit.io/blog/mean-vs-mern-the-ultimate-comparison-2022/> (accessed Apr. 29, 2023).
- [3] 'MongoDB: The Developer Data Platform | MongoDB'. <https://www.mongodb.com/> (accessed Apr. 24, 2023).
- [4] 'Express - Node.js web application framework'. <https://expressjs.com/> (accessed Apr. 24, 2023).
- [5] 'React'. <https://react.dev/> (accessed Apr. 24, 2023).
- [6] 'Node.js'. <https://nodejs.org/en> (accessed Apr. 24, 2023).

- [7] 'MUI: The React component library you always wanted'. <https://mui.com/> (accessed Apr. 24, 2023).
- [8] 'Auth0: Secure access for everyone. But not just anyone.' <https://auth0.com/> (accessed Apr. 29, 2023).
- [9] 'API Hub - Free Public & Open Rest APIs | Rapid'. <https://rapidapi.com/hub> (accessed Apr. 29, 2023).
- [10] 'ExerciseDB API Documentation (justin-WFnsXH_t6) | RapidAPI'.
https://rapidapi.com/justin-WFnsXH_t6/api/exercisedb (accessed Apr. 29, 2023).
- [11] 'Collaboration software for software, IT and business teams'.
<https://www.atlassian.com/> (accessed Apr. 29, 2023).
- [12] 'Jira | Issue & Project Tracking Software | Atlassian'.
<https://www.atlassian.com/software/jira> (accessed Apr. 29, 2023).
- [13] 'Bitbucket | Git solution for teams using Jira'. <https://bitbucket.org/product> (accessed May 03, 2023).
- [14] 'Auth0 React SDK Quickstarts: Call an API'.
<https://auth0.com/docs/quickstart/spa/react/02-calling-an-api> (accessed Apr. 30, 2023).
- [15] 'javascript - Create a Mongoose Schema of type Array of Maps - Stack Overflow'.
<https://stackoverflow.com/questions/54122417/create-a-mongoose-schema-of-type-array-of-maps> (accessed Apr. 30, 2023).
- [16] 'Mongoose v7.1.0: SchemaTypes'.
<https://mongoosejs.com/docs/schematypes.html#maps> (accessed Apr. 30, 2023).
- [17] 'Authentication API Explorer'.
<https://auth0.com/docs/api/authentication?javascript#introduction> (accessed Apr. 30, 2023).

- [18] 'Sarcopenia (Muscle Loss): Symptoms & Causes'.
<https://my.clevelandclinic.org/health/diseases/23167-sarcopenia> (accessed May 01, 2023).
- [19] 'How can strength training build healthier bodies as we age? | National Institute on Aging'. <https://www.nia.nih.gov/news/how-can-strength-training-build-healthier-bodies-we-age> (accessed May 01, 2023).
- [20] G. Kennedy, R. J. Hardman, H. MacPherson, A. B. Scholey, and A. Pipingas, 'How Does Exercise Reduce the Rate of Age-Associated Cognitive Decline? A Review of Potential Mechanisms', *J Alzheimers Dis*, vol. 55, no. 1, pp. 1–18, 2017, doi: 10.3233/JAD-160665.