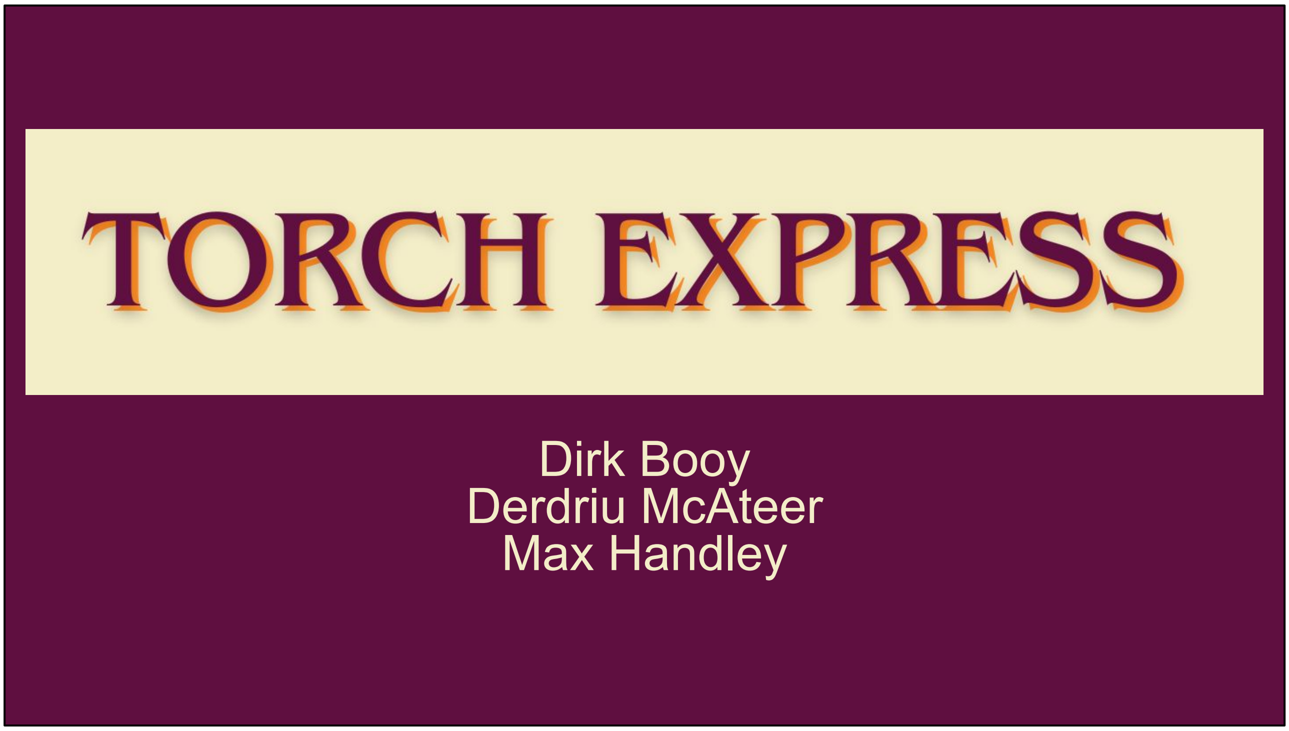
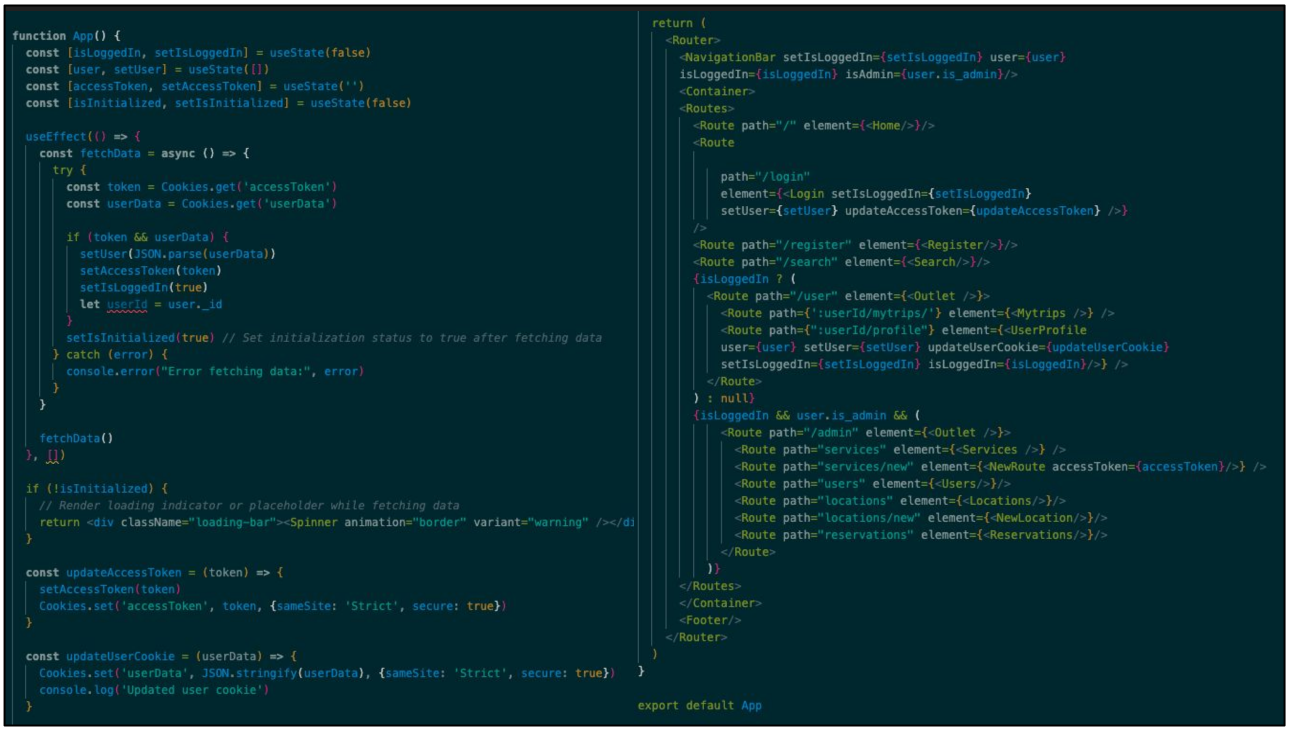


TORCH EXPRESS

Dirk Booy
Derdriu McAteer
Max Handley

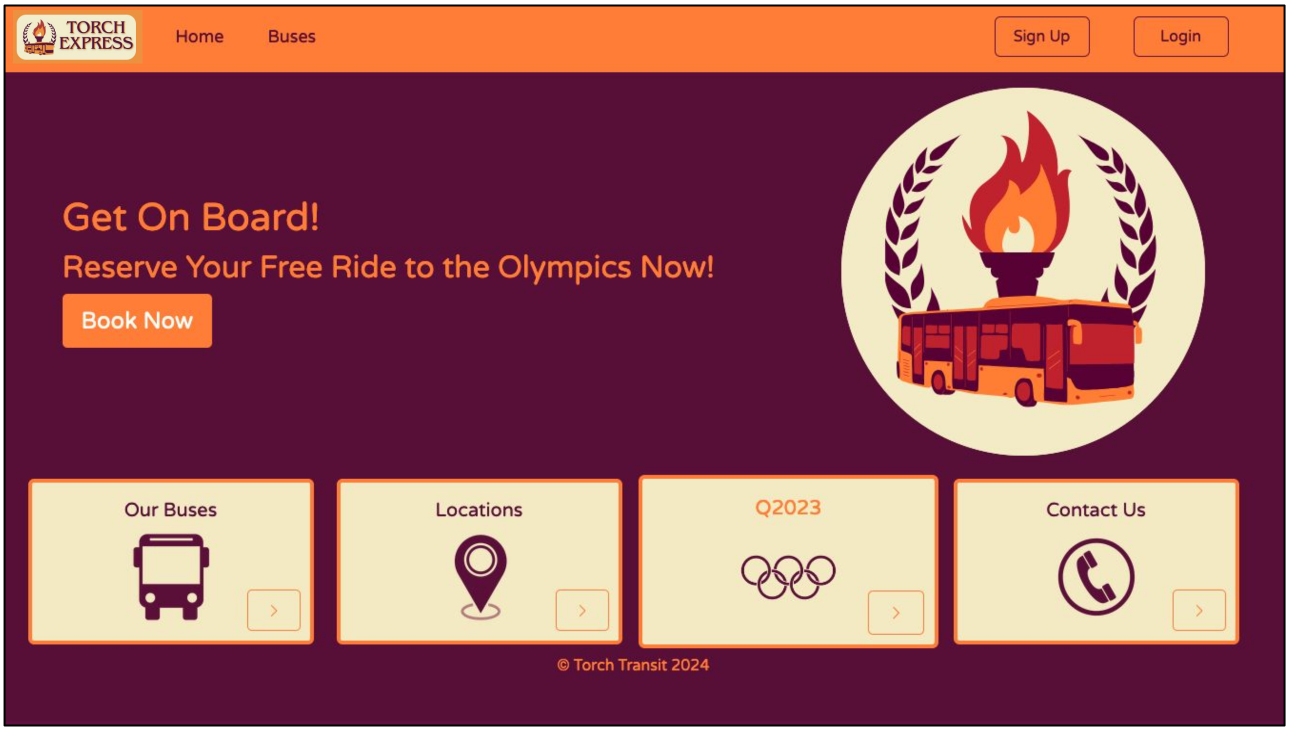
- Your app is a transportation service platform designed to connect users with available transportation services based on their location and travel date.
- **Problem**: Many users struggle to find convenient and reliable transportation options for their travel needs, especially in urban areas or when planning trips in advance.
- **Solution**: Your app streamlines the process of searching for transportation services by providing a user-friendly interface where users can specify their pickup and dropoff locations, as well as their desired travel date, to find available options efficiently.



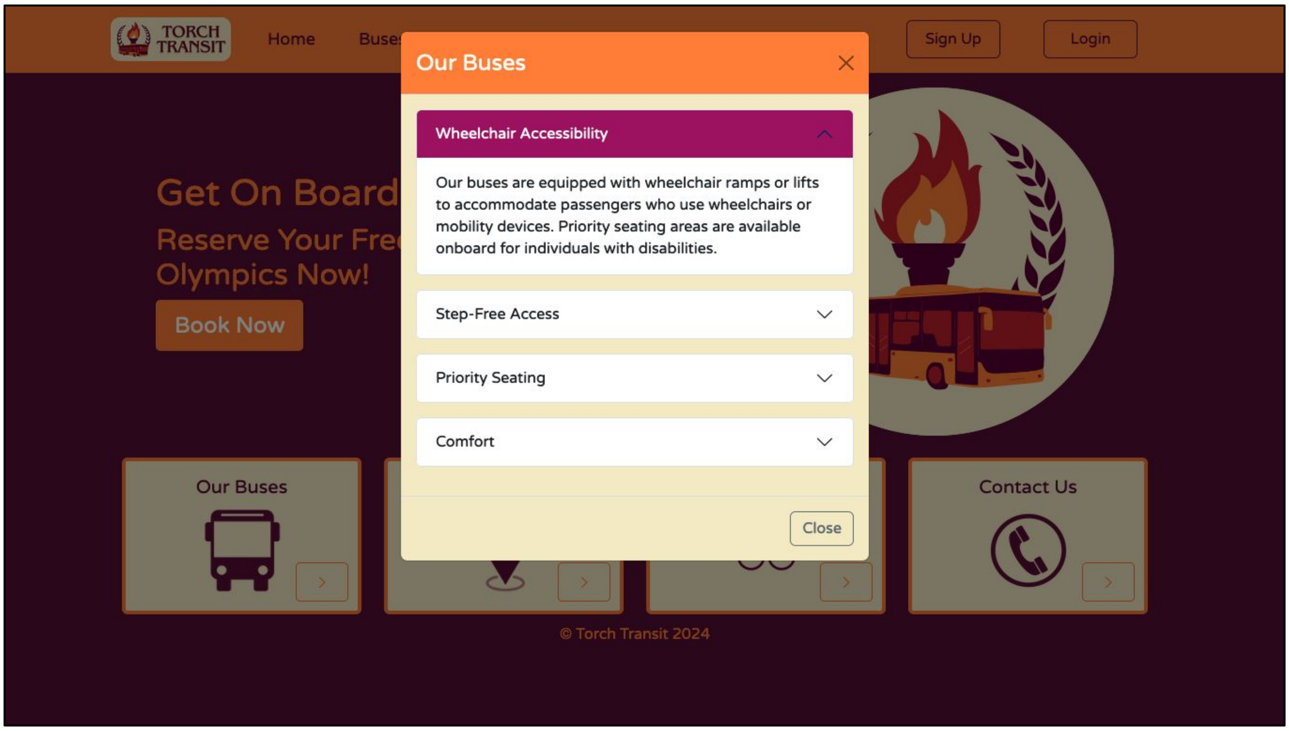
```jsx
function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false)
  const [user, setUser] = useState([])
  const [accessToken, setAccessToken] = useState('')
  const [isInitialized, setIsInitialized] = useState(false)

  useEffect(() => {
    const fetchData = async () => {
      try {
        const token = Cookies.get('accessToken')
        const userData = Cookies.get('userData')

        if (token && userData) {
          setUser(JSON.parse(userData))
          setAccessToken(token)
          setIsLoggedIn(true)
          let userId = user._id
        }
        setIsInitialized(true) // Set initialization status to true after fetching data
      } catch (error) {
        console.error("Error fetching data:", error)
      }
    }

    fetchData()
  }, [])

  if (!isInitialized) {
    // Render loading indicator or placeholder while fetching data
    return <div className="loading-bar"><Spinner animation="border" variant="warning" /></di
  }

  const updateAccessToken = (token) => {
    setAccessToken(token)
    Cookies.set('accessToken', token, {sameSite: 'Strict', secure: true})
  }

  const updateUserCookie = (userData) => {
    Cookies.set('userData', JSON.stringify(userData), {sameSite: 'Strict', secure: true})
    console.log('Updated user cookie')
  }
```

```jsx
  return (
    <Router>
      <NavigationBar setIsLoggedIn={setIsLoggedIn} user={user}
      isLoggedIn={isLoggedIn} isAdmin={user.is_admin}/>
      <Container>
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route


          path="/login"
          element={<Login setIsLoggedIn={setIsLoggedIn}
          setUser={setUser} updateAccessToken={updateAccessToken} />}
        />
        <Route path="/register" element={<Register/>}/>
        <Route path="/search" element={<Search/>}/>
        {isLoggedIn ? (
          <Route path="/user" element={<Outlet />}>
            <Route path={':userId/mytrips/'} element={<Mytrips />} />
            <Route path={":userId/profile"} element={<UserProfile
            user={user} setUser={setUser} updateUserCookie={updateUserCookie}
            setIsLoggedIn={setIsLoggedIn} isLoggedIn={isLoggedIn}/>} />
          </Route>
        ) : null}
        {isLoggedIn && user.is_admin && (
            <Route path="/admin" element={<Outlet />}>
              <Route path="services" element={<Services />} />
              <Route path="services/new" element={<NewRoute accessToken={accessToken}/>} />
              <Route path="users" element={<Users/>}/>
              <Route path="locations" element={<Locations/>}/>
              <Route path="locations/new" element={<NewLocation/>}/>
              <Route path="reservations" element={<Reservations/>}/>
            </Route>
          )}
      </Routes>
      </Container>
      <Footer/>
    </Router>
  )
}

export default App
```

- The App component is the main page displaying what is seen in the Single Page Application (SPA)
- It sets up routing using react-router-dom to handle navigation between different pages.
- The useEffect hook is used to fetch user data and access tokens from cookies when the component mounts, if the user is logged in.
- This component renders the base of what the SPA has on every route, a navigation bar, the main content area, and a footer.
- The content of the main area is determined by the current route, which is specified using Route components.
- Conditional rendering is used on whether the user is logged in and their role (admin or regular user) by the use of states, isLoggedIn and information put in the User state by cookies.
- Initial use of the application, the state isLoggedIn is set to false, so therefore Login and Register links are visible, as well as a home route and Buses, displaying available buses.



- The landing page of the application is displayed when a non-logged-in user visits the website.
- The color palette used in the final application is supposed to mimic the colours of a deep burning flame.
- At the top the Navigation Bar is visible with links to to the route (this current page) being the app logo, and the home button
- Conditionally rendered are the login and signup links in the NavBar.
- It consists of a banner section with a button to book a ride to the Olympics.
- Below the banner, there are four cards representing different features of the application: Buses, Locations, Q2023, and Contact Us.
- Clicking on the "Buses" card opens a modal with information about the buses.
- Clicking on the "Locations" card opens a modal with information about various locations.
- Clicking on the "Q2023" card redirects the user to an external link related to the Olympics.
- Clicking on the "Contact Us" card opens a modal with contact information.



Our Buses

Wheelchair Accessibility

Our buses are equipped with wheelchair ramps or lifts to accommodate passengers who use wheelchairs or mobility devices. Priority seating areas are available onboard for individuals with disabilities.

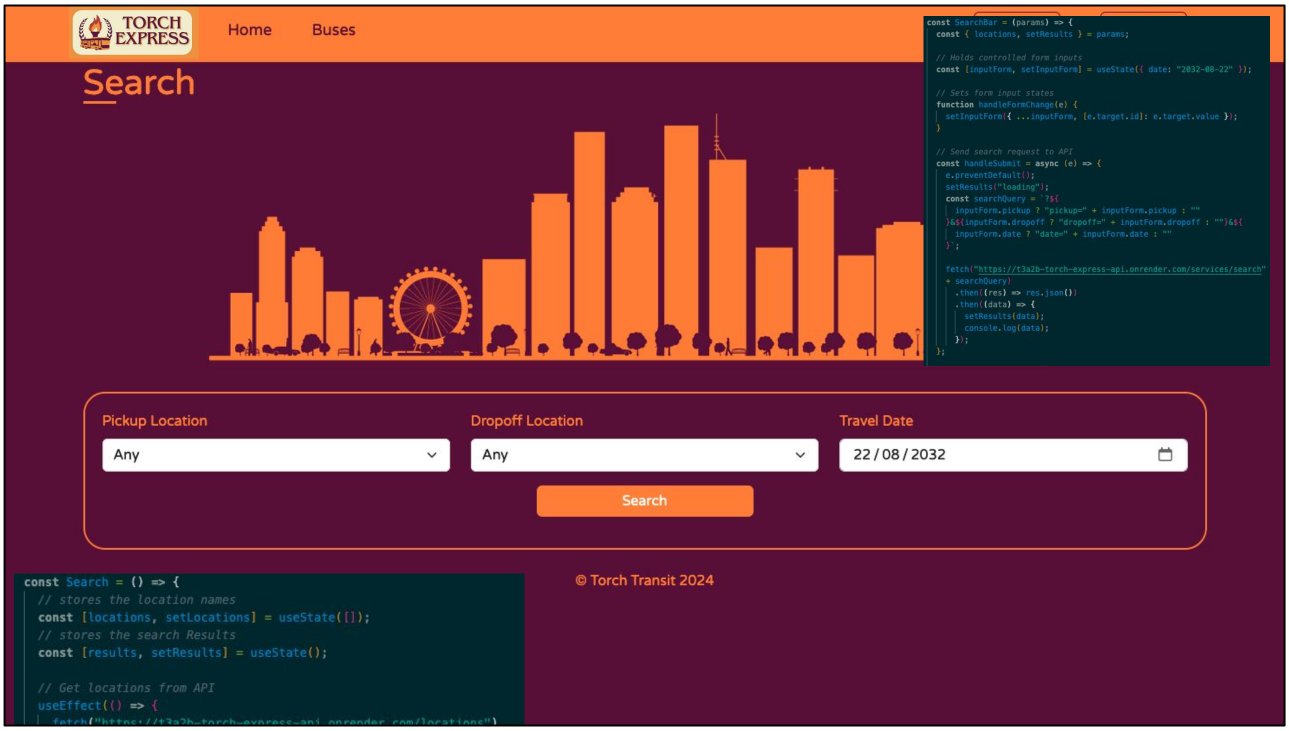
Step-Free Access

Priority Seating

Comfort

Close

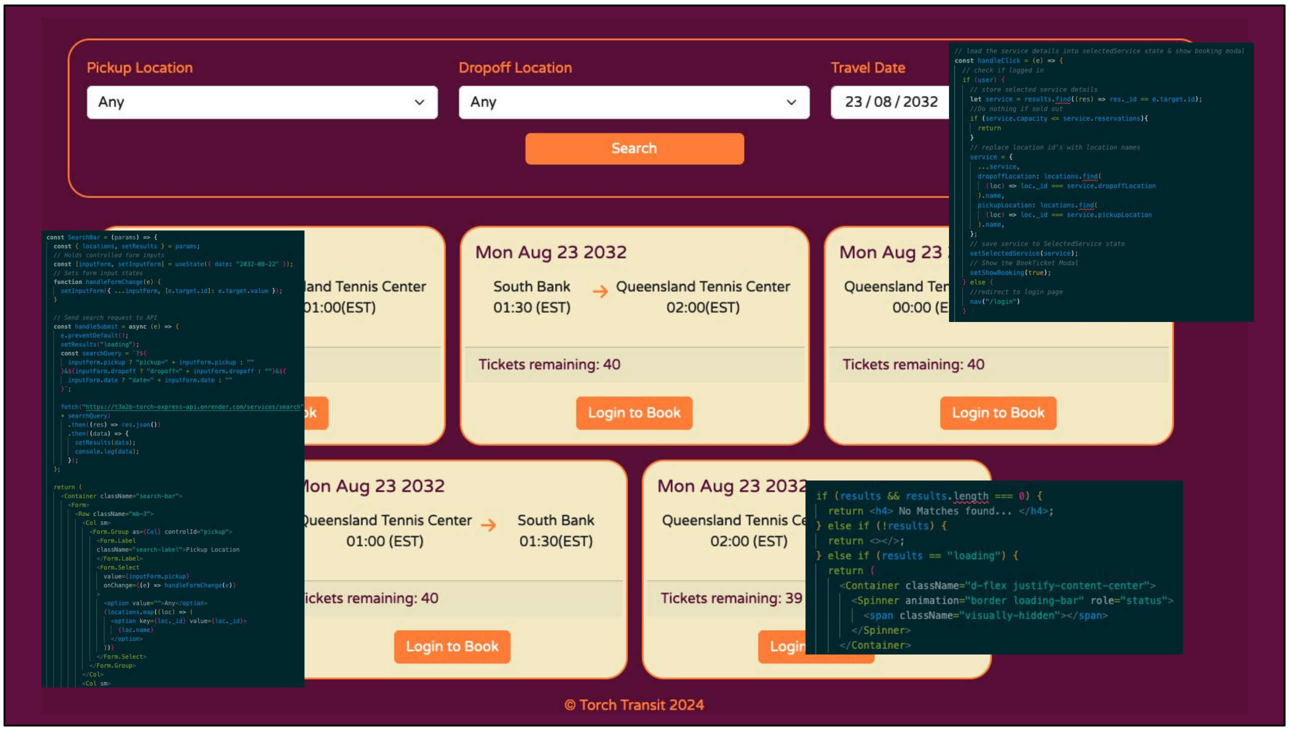
- This exemplifies the Our Buses modal being interacted with, displaying information about the bus and it’s seating.
- All models have information about the company and services, bar the link to the olympics



```
const SearchBar = (params) => {
  const { locations, setResults } = params;

  // Holds controlled form inputs
  const [inputForm, setInputForm] = useState({ date: "2032-08-22" });

  // Sets form input states
  function handleFormChange(e) {
    setInputForm({ ...inputForm, [e.target.id]: e.target.value });
  }

  // Send search request to API
  const handleSubmit = async (e) => {
    e.preventDefault();
    setResults("loading");
    const searchQuery = `?${
      inputForm.pickup ? "pickup=" + inputForm.pickup : ""
    }&${inputForm.dropoff ? "dropoff=" + inputForm.dropoff : ""}&${
      inputForm.date ? "date=" + inputForm.date : ""
    }`;

    fetch("https://t3a2b-torch-express-api.onrender.com/services/search"
    + searchQuery)
      .then((res) => res.json())
      .then((data) => {
        setResults(data);
        console.log(data);
      });
};
```

```
const Search = () => {
  // stores the location names
  const [locations, setLocations] = useState([]);
  // stores the search Results
  const [results, setResults] = useState();

  // Get locations from API
  useEffect(() => {
    fetch("https://t3a2b-torch-express-api.onrender.com/locations")
```

- The Search page is a crucial component of our application, allowing users to find relevant information based on their preferences
- This is the main crux of the application, an interface for users to search for available services.
- We utilise the useEffect hook to fetch location data from our API endpoint when the component mounts.
- The fetched location data is stored in the locations state using the useState hook.
- The Search page includes a SearchBar component responsible for rendering the search form.
- Users can select their pickup and dropoff locations from dropdown menus populated with the fetched location data and specify their desired travel date using a date picker input.
- The SearchBar component handles form inputs through controlled components, updating the inputForm state accordingly.



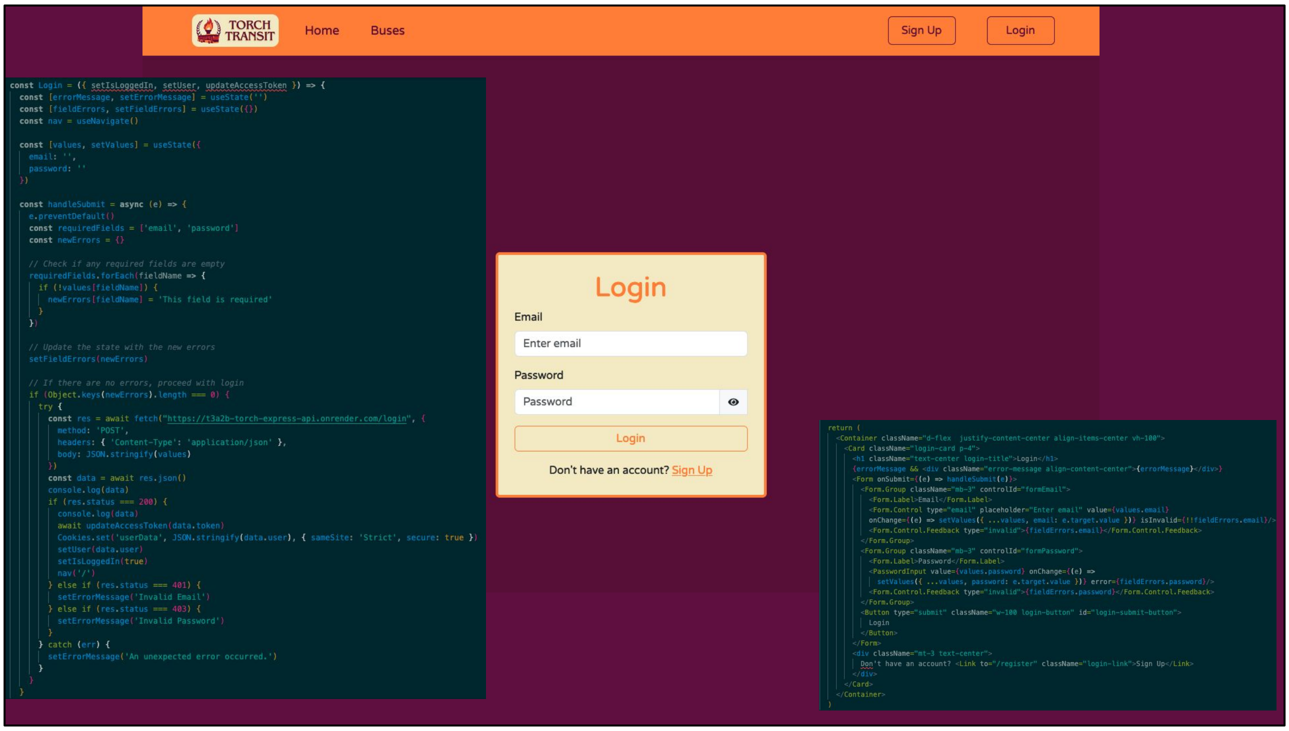
```
const SearchBar = (params) => {
  const { locations, setResults } = params;
  // Holds controlled form inputs
  const [inputForm, setInputForm] = useState({ date: "2032-08-22" });
  // Sets form input states
  function handleFormChange(e) {
    setInputForm({ ...inputForm, [e.target.id]: e.target.value });
  }

  // Send search request to API
  const handleSubmit = async (e) => {
    e.preventDefault();
    setResults("loading");
    const searchQuery = `?${
      inputForm.pickup ? "pickup=" + inputForm.pickup : ""
    }&${inputForm.dropoff ? "dropoff=" + inputForm.dropoff : ""}&${
      inputForm.date ? "date=" + inputForm.date : ""
    }`;

    fetch("https://t3a2b-torch-express-api.onrender.com/services/search"
    + searchQuery)
      .then((res) => res.json())
      .then((data) => {
        setResults(data);
        console.log(data);
      });
  };

  return (
    <Container className="search-bar">
      <Form>
        <Row className="mb-3">
          <Col sm>
            <Form.Group as={Col} controlId="pickup">
              <Form.Label
              className="search-label">Pickup Location
              </Form.Label>
              <Form.Select
                value={inputForm.pickup}
                onChange={(e) => handleFormChange(e)}
              >
                <option value="">Any</option>
                {locations.map((loc) => (
                  <option key={loc._id} value={loc._id}>
                    {loc.name}
                  </option>
                ))}
              </Form.Select>
            </Form.Group>
          </Col>
          <Col sm>
```

```
// load the service details into selectedService state & show booking modal
const handleClick = (e) => {
  // check if logged in
  if (user) {
    // store selected service details
    let service = results.find((res) => res._id == e.target.id);
    //Do nothing if sold out
    if (service.capacity <= service.reservations){
      return
    }
    // replace location id's with location names
    service = {
      ...service,
      dropoffLocation: locations.find(
        (loc) => loc._id === service.dropoffLocation
      ).name,
      pickupLocation: locations.find(
        (loc) => loc._id === service.pickupLocation
      ).name,
    };
    // save service to SelectedService state
    setSelectedService(service);
    // Show the BookTicket Modal
    setShowBooking(true);
  } else {
    //redirect to login page
    nav("/login")
  }
```

```
if (results && results.length === 0) {
  return <h4> No Matches found... </h4>;
} else if (!results) {
  return <></>;
} else if (results == "loading") {
  return (
    <Container className="d-flex justify-content-center">
      <Spinner animation="border loading-bar" role="status">
        <span className="visually-hidden"></span>
      </Spinner>
    </Container>
```

Displaying Search Results

- When users submit the search form, the handleSubmit function constructs a search query based on the selected options in the search bar
- A get request is sent to the route collection, and the logic in handleSubmit using the searchQuery variable filters the data.
- The filtered results are displayed on the page using the SearchResults component.
- As users interact with the search form, the displayed results dynamically update to reflect their search criteria.
- If there are no results, a heading with no matches is rendered.
- The “book now” button is set to a login link if the the cookie isn’t containing the user details, the logic handled in the pictured handleClick function.

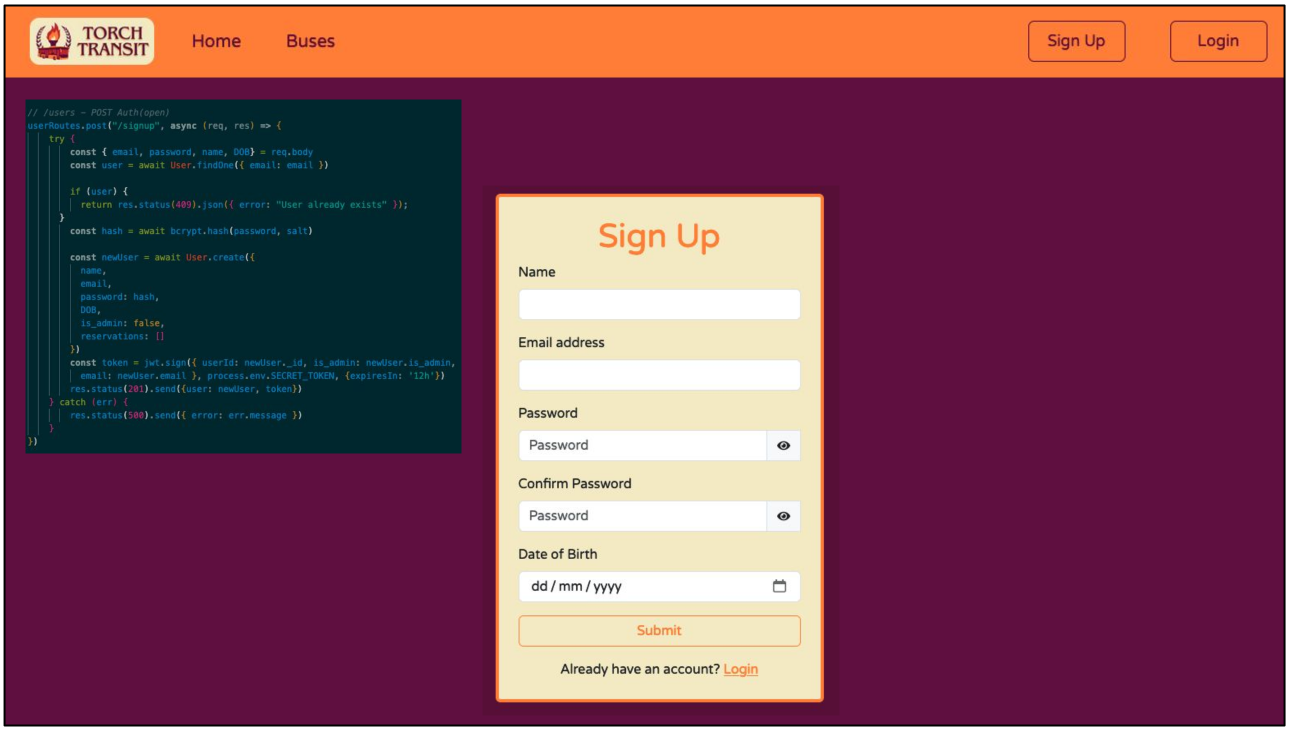


```
const Login = ({ setIsLoggedIn, setUser, updateAccessToken }) => {
  const [errorMessage, setErrorMessage] = useState('')
  const [fieldErrors, setFieldErrors] = useState({})
  const nav = useNavigate()

  const [values, setValues] = useState({
    email: '',
    password: ''
  })

  const handleSubmit = async (e) => {
    e.preventDefault()
    const requiredFields = ['email', 'password']
    const newErrors = {}

    // Check if any required fields are empty
    requiredFields.forEach(fieldName => {
      if (!values[fieldName]) {
        newErrors[fieldName] = 'This field is required'
      }
    })

    // Update the state with the new errors
    setFieldErrors(newErrors)

    // If there are no errors, proceed with login
    if (Object.keys(newErrors).length === 0) {
      try {
        const res = await fetch("https://t3a2b-torch-express-api.onrender.com/login", {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(values)
        })
        const data = await res.json()
        console.log(data)
        if (res.status === 200) {
          console.log(data)
          await updateAccessToken(data.token)
          Cookies.set('userData', JSON.stringify(data.user), { sameSite: 'Strict', secure: true })
          setUser(data.user)
          setIsLoggedIn(true)
          nav('/')
        } else if (res.status === 401) {
          setErrorMessage('Invalid Email')
        } else if (res.status === 403) {
          setErrorMessage('Invalid Password')
        }
      } catch (err) {
        setErrorMessage('An unexpected error occurred.')
      }
    }
  }
```

```
return (
  <Container className="d-flex  justify-content-center align-items-center vh-100">
    <Card className="login-card p-4">
      <h1 className="text-center login-title">Login</h1>
      {errorMessage && <div className="error-message align-content-center">{errorMessage}</div>}
      <Form onSubmit={(e) => handleSubmit(e)}>
        <Form.Group className="mb-3" controlId="formEmail">
          <Form.Label>Email</Form.Label>
          <Form.Control type="email" placeholder="Enter email" value={values.email}
          onChange={(e) => setValues({ ...values, email: e.target.value })} isInvalid={!!fieldErrors.email}/>
          <Form.Control.Feedback type="invalid">{fieldErrors.email}</Form.Control.Feedback>
        </Form.Group>
        <Form.Group className="mb-3" controlId="formPassword">
          <Form.Label>Password</Form.Label>
          <PasswordInput value={values.password} onChange={(e) =>
            setValues({ ...values, password: e.target.value })} error={fieldErrors.password}/>
          <Form.Control.Feedback type="invalid">{fieldErrors.password}</Form.Control.Feedback>
        </Form.Group>
        <Button type="submit" className="w-100 login-button" id="login-submit-button">
          Login
        </Button>
      </Form>
      <div className="mt-3 text-center">
        Don't have an account? <Link to="/register" className="login-link">Sign Up</Link>
      </div>
    </Card>
  </Container>
)
```

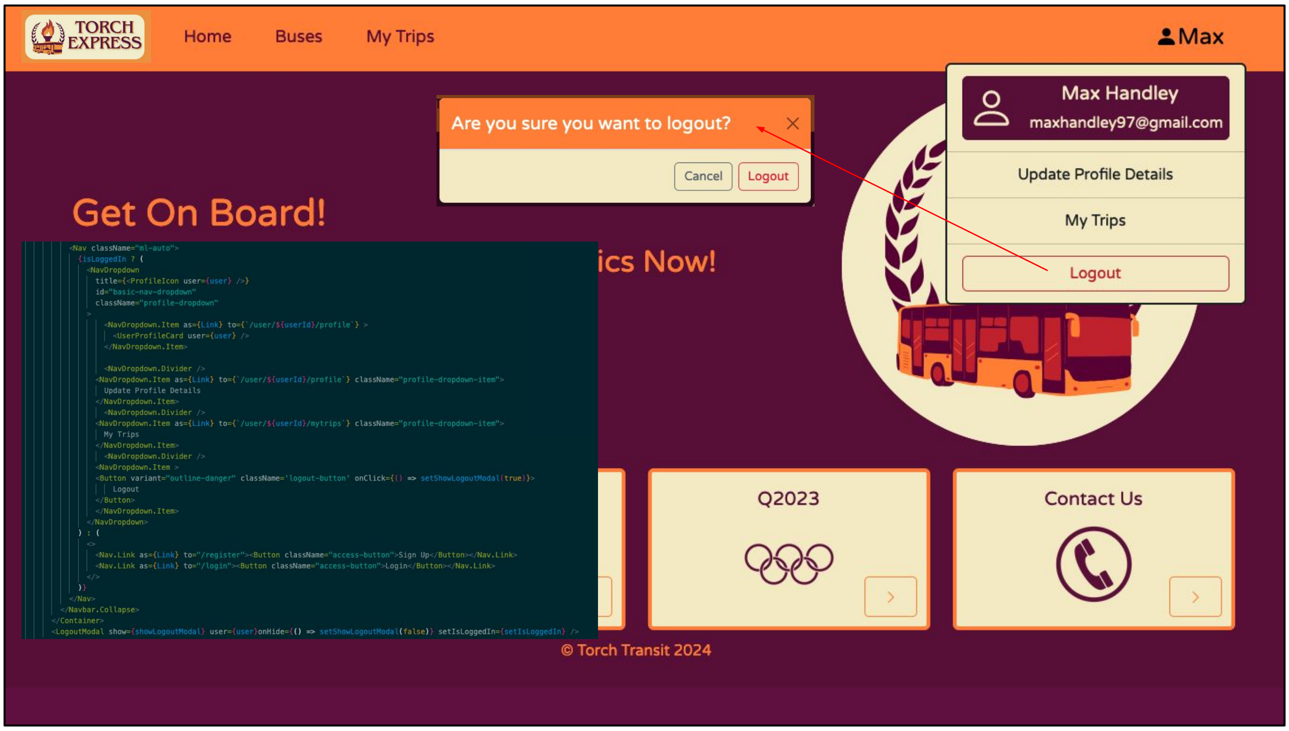
Login Page

- A form with two input fields: one for email and one for password, stored in requiredFields state.
- Error messages are held in a state newErrors, to dynamically to provide feedback on input errors.
- Upon form submit, pressing the login button, credentials are sent to the server for authentication.
- If invalid or password hashes do not match, error messages guide users on how to proceed
- If password hashes match bearer token is return, user details and jwt are stored in a cookie.
- If credentials are valid, users are redirected to the main dashboard.
- Users are prompted to the registration page if they don't have an account yet.



```
// /users - POST Auth(open)
userRoutes.post("/signup", async (req, res) => {
    try {
        const { email, password, name, DOB} = req.body
        const user = await User.findOne({ email: email })

        if (user) {
          return res.status(409).json({ error: "User already exists" });
      }
        const hash = await bcrypt.hash(password, salt)

        const newUser = await User.create({
          name,
          email,
          password: hash,
          DOB,
          is_admin: false,
          reservations: []
        })
        const token = jwt.sign({ userId: newUser._id, is_admin: newUser.is_admin,
          email: newUser.email }, process.env.SECRET_TOKEN, {expiresIn: '12h'})
        res.status(201).send({user: newUser, token})
    } catch (err) {
        res.status(500).send({ error: err.message })
    }
})
```

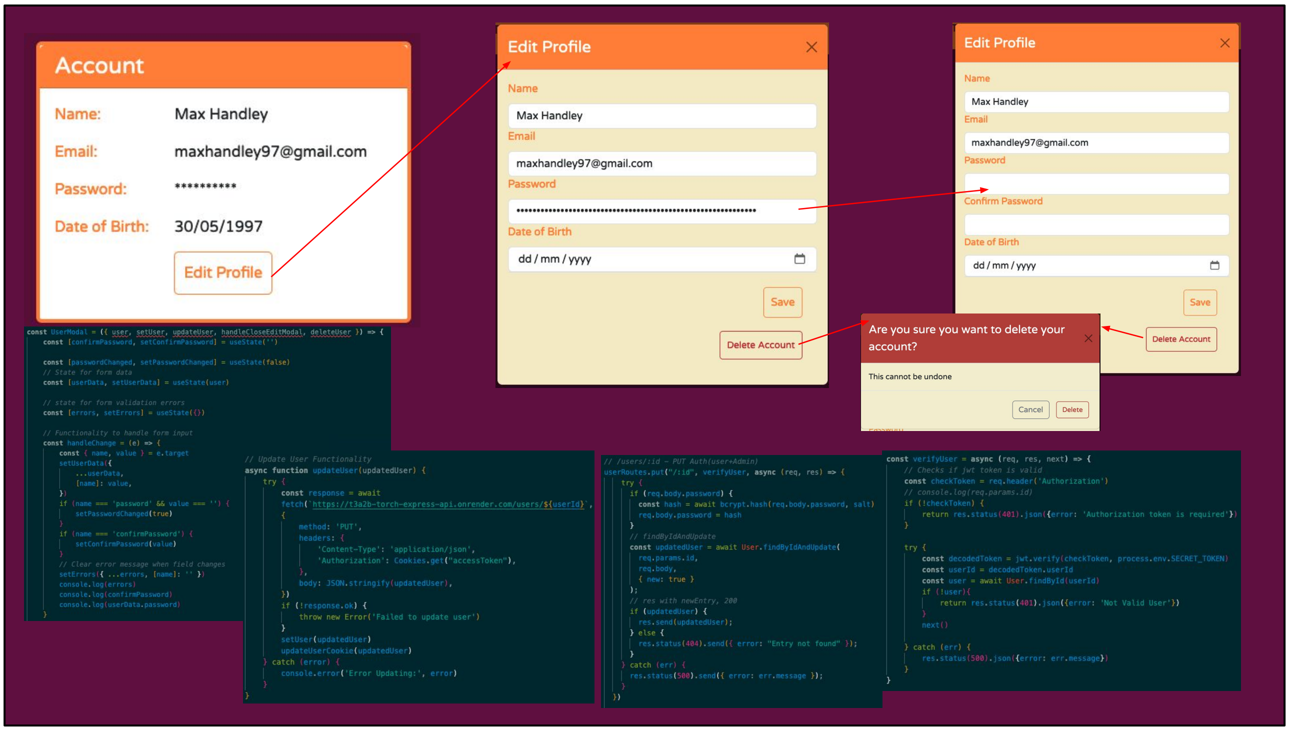
- This is the registration process on our application uses the back end /signup route.
- When users arrive at the registration page, they're presented with a form containing several fields, including Name, Email Address, Password, Confirm Password, and Date of Birth.
- These fields are held in the values state, two password inputs must match to be able to create an account.
- As users fill out these fields, we validate their inputs in real-time, providing immediate feedback if any required fields are left empty or if the passwords don't match.
- Once all required information is provided and validated, users submit the form.
- Further validation occurs with the backend, if email is being used an error is thrown and displayed to user.
- Before sending, their data is validated and the password is hashed, then the data is sent to our server.
- If the registration is successful, users are redirected to the login page to access their new account.



```
          <Nav className="ml-auto">
            {isLoggedIn ? (
              <NavDropdown
                title={<ProfileIcon user={user} />}
                id="basic-nav-dropdown"
                className="profile-dropdown"
              >
                  <NavDropdown.Item as={Link} to={`/user/${userId}/profile`} >
                    <UserProfileCard user={user} />
                  </NavDropdown.Item>

                  <NavDropdown.Divider />
                <NavDropdown.Item as={Link} to={`/user/${userId}/profile`} className="profile-dropdown-item">
                  Update Profile Details
                </NavDropdown.Item>
                  <NavDropdown.Divider />
                <NavDropdown.Item as={Link} to={`/user/${userId}/mytrips`} className="profile-dropdown-item">
                  My Trips
                </NavDropdown.Item>
                  <NavDropdown.Divider />
                <NavDropdown.Item >
                <Button variant="outline-danger" className='logout-button' onClick={() => setShowLogoutModal(true)}>
                    Logout
                </Button>
                </NavDropdown.Item>
              </NavDropdown>
            ) : (
              <>
                <Nav.Link as={Link} to="/register"><Button className="access-button">Sign Up</Button></Nav.Link>
                <Nav.Link as={Link} to="/login"><Button className="access-button">Login</Button></Nav.Link>
              </>
            )}
          </Nav>
        </Navbar.Collapse>
      </Container>
      <LogoutModal show={showLogoutModal} user={user}onHide={() => setShowLogoutModal(false)} setIsLoggedIn={setIsLoggedIn} />
```

Upon Login

- Once logged in the user data is used in the navbar to conditionally render a navbar with myTrips and User details.
- The user dropdown displays account information in a card, and links to the account details.
- The update Profile navigates to the users' profile details in order to edit them.
- My Trips navigates to the user's reservations.
- The logout button, when clicked displays the pictured modal in order for the user to continue the log out process.
- Once the logout within the modal is clicked, this triggers cookies to be emptied and isLoggedIn to be set back to false and navigation back to the login component.
- When Update Profile Details or the User Profile card are clicked, the user is redirected to UserProfile



Account

Name: Max Handley
Email: maxhandley97@gmail.com
Password: **********
Date of Birth: 30/05/1997

Edit Profile

Name: Max Handley
Email: maxhandley97@gmail.com
Password: ••
Date of Birth: dd / mm / yyyy

Edit Profile

Name: Max Handley
Email: maxhandley97@gmail.com
Password:
Confirm Password:
Date of Birth: dd / mm / yyyy

Are you sure you want to delete your account?

This cannot be undone

```javascript
const UserModal = ({ user, setUser, updateUser, handleCloseEditModal, deleteUser }) => {
    const [confirmPassword, setConfirmPassword] = useState('')

    const [passwordChanged, setPasswordChanged] = useState(false)
    // State for form data
    const [userData, setUserData] = useState(user)

    // state for form validation errors
    const [errors, setErrors] = useState({})

    // Functionality to handle form input
    const handleChange = (e) => {
        const { name, value } = e.target
        setUserData({
            ...userData,
            [name]: value,
        })
        if (name === 'password' && value === '') {
            setPasswordChanged(true)
        }
        if (name === 'confirmPassword') {
            setConfirmPassword(value)
        }
        // Clear error message when field changes
        setErrors({ ...errors, [name]: '' })
        console.log(errors)
        console.log(confirmPassword)
        console.log(userData.password)
    }
```

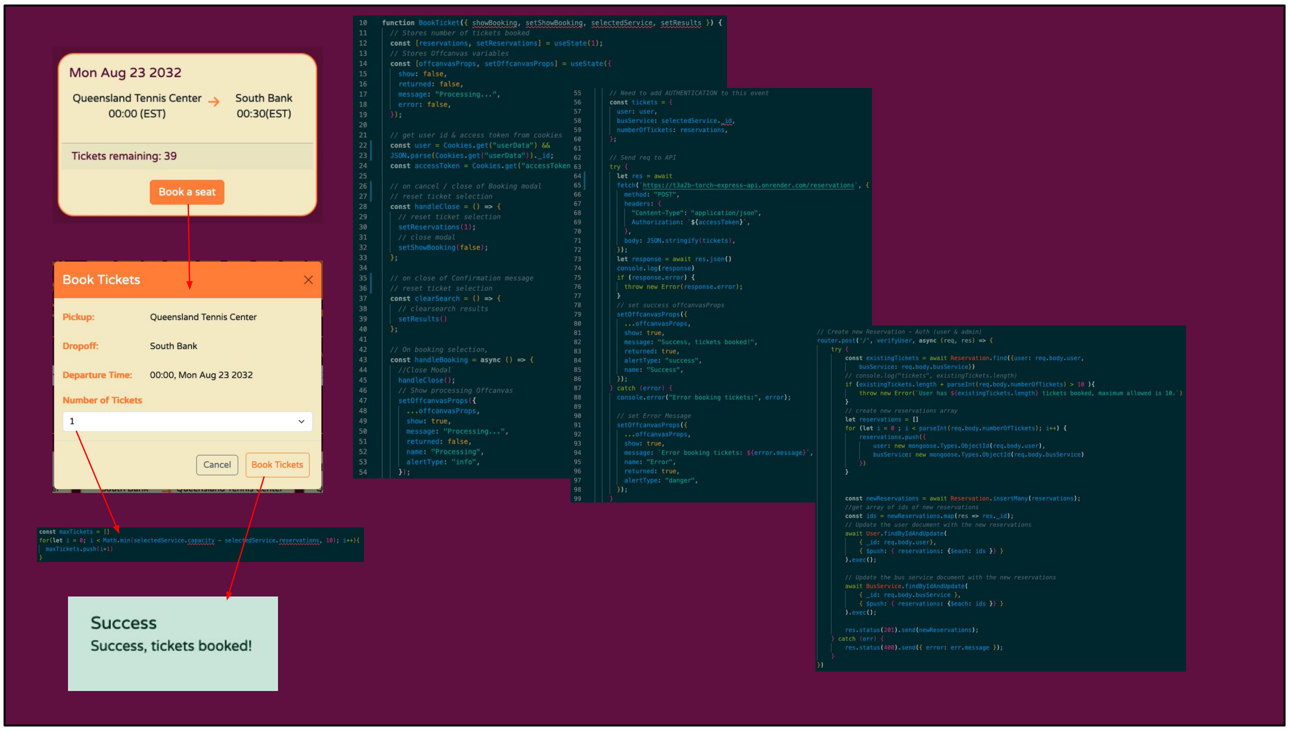
```javascript
// Update User Functionality
async function updateUser(updatedUser) {
    try {
        const response = await
        fetch(`https://t3a2b-torch-express-api.onrender.com/users/${userId}`,
        {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': Cookies.get("accessToken"),
            },
            body: JSON.stringify(updatedUser),
        })
        if (!response.ok) {
            throw new Error('Failed to update user')
        }
        setUser(updatedUser)
        updateUserCookie(updatedUser)
    } catch (error) {
        console.error('Error Updating:', error)
    }
}
```

```javascript
// /users/:id - PUT Auth(user+Admin)
userRoutes.put("/:id", verifyUser, async (req, res) => {
    try {
      if (req.body.password) {
        const hash = await bcrypt.hash(req.body.password, salt)
        req.body.password = hash
      }
      // findByIdAndUpdate
      const updatedUser = await User.findByIdAndUpdate(
        req.params.id,
        req.body,
        { new: true }
      );
      // res with newEntry, 200
      if (updatedUser) {
        res.send(updatedUser);
      } else {
        res.status(404).send({ error: "Entry not found" });
      }
    } catch (err) {
      res.status(500).send({ error: err.message });
    }
  })
```

```javascript
const verifyUser = async (req, res, next) => {
    // Checks if jwt token is valid
    const checkToken = req.header('Authorization')
    // console.log(req.params.id)
    if (!checkToken) {
        return res.status(401).json({error: 'Authorization token is required'})
    }

    try {
        const decodedToken = jwt.verify(checkToken, process.env.SECRET_TOKEN)
        const userId = decodedToken.userId
        const user = await User.findById(userId)
        if (!user){
            return res.status(401).json({error: 'Not Valid User'})
        }
        next()

    } catch (err) {
        res.status(500).json({error: err.message})
    }
}
```

- On the left of the slide, the user profile is displayed, which is found at the route "user/65e056315af9e941a7091a9c/profile"
- This page serves as a simple view for users to access their account details and make necessary edits.
- This simple page displays the user account details and an edit profile button, which when clicked opens a modal.
- Users can view and modify details being their name, email, password, and date of birth.
- Access and manipulation is only possible by the requests having the authorisation header, being the JWT token created at login.
- This is guided by the backend logic, exemplified in the bottom right corner of the slide
- The verifyUser function ensures the owner user (currently logged) is the only user capable of changing these credentials/
- The user profile is displayed by the UserProfile component, which handles the logic of sending delete and put requests to the server and database.
- Props from the parent "App" component are used update user details live on the front end as well as changing them in the database. The user modal.
- A password confirmation field dynamically appears if the user decides to change their password, ensuring intention.
- Form validation alerts users of any input errors, including missing fields if edited and password matching.
- Upon submission of the form, user data is updated on the server and client

- sides.
- Account deletion is a two step process with a pop out modal, to ensure user intention.
- Once delete modal is submitted, logic similar to the displayed put logic ensures only the owner user (or admin) can send delete requests to the DB.



Book Tickets

Pickup:	Queensland Tennis Center
Dropoff:	South Bank
Departure Time:	00:00, Mon Aug 23 2032
Number of Tickets	1

```javascript
function BookTicket({ showBooking, setShowBooking, selectedService, setResults }) {
  // Stores number of tickets booked
  const [reservations, setReservations] = useState(1);
  // Stores Offcanvas variables
  const [offcanvasProps, setOffcanvasProps] = useState({
    show: false,
    returned: false,
    message: "Processing...",
    error: false,
  });

  // get user id & access token from cookies
  const user = Cookies.get("userData") &&
  JSON.parse(Cookies.get("userData"))._id;
  const accessToken = Cookies.get("accessToken

  // on cancel / close of Booking modal
  // reset ticket selection
  const handleClose = () => {
    // reset ticket selection
    setReservations(1);
    // close modal
    setShowBooking(false);
  };

  // on close of Confirmation message
  // reset ticket selection
  const clearSearch = () => {
    // clearsearch results
    setResults()
  };

  // On booking selection,
  const handleBooking = async () => {
    //Close Modal
    handleClose();
    // Show processing Offcanvas
    setOffcanvasProps({
      ...offcanvasProps,
      show: true,
      message: "Processing...",
      returned: false,
      name: "Processing",
      alertType: "info",
    });
```

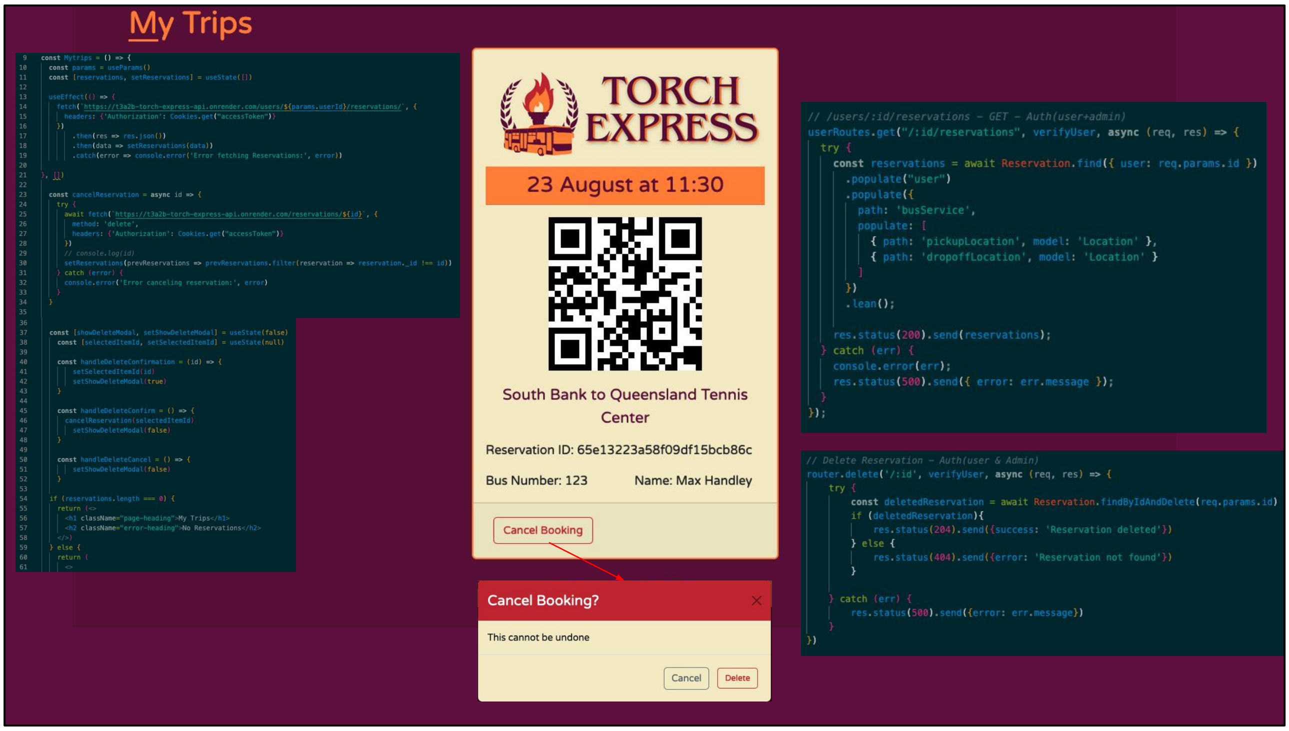
```javascript
    // Need to add AUTHENTICATION to this event
    const tickets = {
      user: user,
      busService: selectedService._id,
      numberOfTickets: reservations,
    };

    // Send req to API
    try {
      let res = await
      fetch(`https://t3a2b-torch-express-api.onrender.com/reservations`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `${accessToken}`,
        },
        body: JSON.stringify(tickets),
      });
      let response = await res.json()
      console.log(response)
      if (response.error) {
        throw new Error(response.error);
      }
      // set success offcanvasProps
      setOffcanvasProps({
        ...offcanvasProps,
        show: true,
        message: "Success, tickets booked!",
        returned: true,
        alertType: "success",
        name: "Success",
      });
    } catch (error) {
      console.error("Error booking tickets:", error);

      // set Error Message
      setOffcanvasProps({
        ...offcanvasProps,
        show: true,
        message: `Error booking tickets: ${error.message}`,
        name: "Error",
        returned: true,
        alertType: "danger",
      });
    }
```

```javascript
const maxTickets = []
for(let i = 0; i < Math.min(selectedService.capacity - selectedService.reservations, 10); i++){
  maxTickets.push(i+1)
}
```

```javascript
// Create new Reservation - Auth (user & admin)
router.post('/', verifyUser, async (req, res) => {
    try {
        const existingTickets = await Reservation.find({user: req.body.user,
            busService: req.body.busService})
        // console.log("tickets", existingTickets.length)
        if (existingTickets.length + parseInt(req.body.numberOfTickets) > 10 ){
            throw new Error(`User has ${existingTickets.length} tickets booked, maximum allowed is 10.`)
        }
        // create new reservations array
        let reservations = []
        for (let i = 0 ; i < parseInt(req.body.numberOfTickets); i++) {
            reservations.push({
                user: new mongoose.Types.ObjectId(req.body.user),
                busService: new mongoose.Types.ObjectId(req.body.busService)
            })
        }


        const newReservations = await Reservation.insertMany(reservations);
        //get array of ids of new reservations
        const ids = newReservations.map(res => res._id);
        // Update the user document with the new reservations
        await User.findByIdAndUpdate(
            { _id: req.body.user},
            { $push: { reservations: {$each: ids }} }
        ).exec();

        // Update the bus service document with the new reservations
        await BusService.findByIdAndUpdate(
            { _id: req.body.busService },
            { $push: { reservations: {$each: ids }} }
        ).exec();

        res.status(201).send(newReservations);
    } catch (err) {
        res.status(400).send({ error: err.message });
    }
})
```

- When logged in user navigates to buses, and uses the search bar (see slide 6) to filter available routes.
- When a user filters available routes using the search bar, a GET request fetches filtered data from the route collection.
- Once the user clicks “Book a seat” on a favourable route, clicking "Book a seat" presents the BookTicket modal component.
- Users select the number of tickets and confirms details such as pickup and drop-off locations.
- Logic displayed by the maxTickets variable ensures the dropdown controlling tickets is limited to the available reservation numbers and already booked tickets.
- Upon clicking "Book Tickets," a reservation object is constructed with user details and sent via a POST request to the API for reservation creation.
- Validation on the server side also ensures limited tickets can be booked on one user account, as well as ensuring the user is authorised implementing VerifyUser.
- Error handling and validation mechanisms on both server and client sides ensure users are alerted if errors occur.
- Upon successful reservation creation, users receive a confirmation message handled by ConfirmationOffCanvas
- This utilises a bootstrap OffCanvas element to create a pop up confirmation,

- making for a smoother UX.



My Trips

```javascript
const Mytrips = () => {
  const params = useParams()
  const [reservations, setReservations] = useState([])

  useEffect(() => {
    fetch(`https://t3a2b-torch-express-api.onrender.com/users/${params.userId}/reservations/`, {
      headers: {'Authorization': Cookies.get("accessToken")}
    })
        .then(res => res.json())
        .then(data => setReservations(data))
        .catch(error => console.error('Error fetching Reservations:', error))

}, [])

  const cancelReservation = async id => {
    try {
      await fetch(`https://t3a2b-torch-express-api.onrender.com/reservations/${id}`, {
        method: 'delete',
        headers: {'Authorization': Cookies.get("accessToken")}
      })
      // console.log(id)
      setReservations(prevReservations => prevReservations.filter(reservation => reservation._id !== id))
    } catch (error) {
      console.error('Error canceling reservation:', error)
    }
  }


  const [showDeleteModal, setShowDeleteModal] = useState(false)
    const [selectedItemId, setSelectedItemId] = useState(null)

    const handleDeleteConfirmation = (id) => {
        setSelectedItemId(id)
        setShowDeleteModal(true)
    }

    const handleDeleteConfirm = () => {
      cancelReservation(selectedItemId)
        setShowDeleteModal(false)
    }

    const handleDeleteCancel = () => {
        setShowDeleteModal(false)
    }

  if (reservations.length === 0) {
    return (<>
      <h1 className="page-heading">My Trips</h1>
      <h2 className="error-heading">No Reservations</h2>
    </>)
  } else {
    return (
      <>
```

23 August at 11:30

South Bank to Queensland Tennis Center

Reservation ID: 65e13223a58f09df15bcb86c

Bus Number: 123 Name: Max Handley

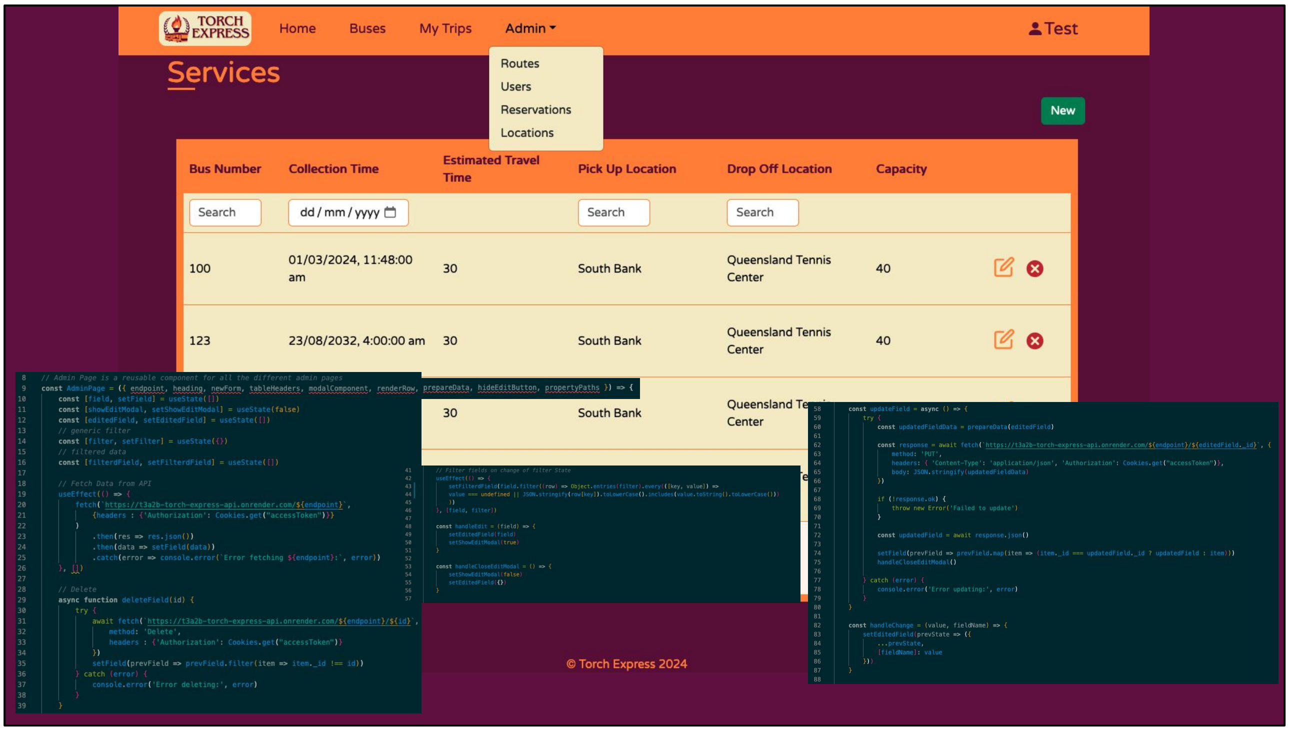
Cancel Booking?

This cannot be undone

```javascript
// /users/:id/reservations - GET - Auth(user+admin)
userRoutes.get("/:id/reservations", verifyUser, async (req, res) => {
  try {
    const reservations = await Reservation.find({ user: req.params.id })
      .populate("user")
      .populate({
        path: 'busService',
        populate: [
          { path: 'pickupLocation', model: 'Location' },
          { path: 'dropoffLocation', model: 'Location' }
        ]
      })
      .lean();

    res.status(200).send(reservations);
  } catch (err) {
    console.error(err);
    res.status(500).send({ error: err.message });
  }
});
```

```javascript
// Delete Reservation - Auth(user & Admin)
router.delete('/:id', verifyUser, async (req, res) => {
  try {
    const deletedReservation = await Reservation.findByIdAndDelete(req.params.id)
    if (deletedReservation){
      res.status(204).send({success: 'Reservation deleted'})
    } else {
      res.status(404).send({error: 'Reservation not found'})
    }

  } catch (err) {
    res.status(500).send({error: err.message})
  }
})
```

- Mytrips serves as the central platform to display the users tickets, applying to restful principles “user/65e13206a58f09df15bcb863/mytrips”
- It employs useEffect with useParams to efficiently when mounting to render all tickets/bookings of the logged in user.
- This is done by sending a Get request to the server, using verifyUser as authorisation.
- As seen on the right side of the slide, a bit of trouble arose when trying to get necessary nested data, which called for nested population as a solution.
- Cancel Booking functionality triggers a pop-out DeleteModal component, providing a confirmation interface for canceling bookings.
- The modal interface offers options to confirm or cancel the cancelation request, ensuring a seamless and user-friendly experience.
- After confirming deletion, application executes a DELETE request to the API endpoint corresponding to the specific reservation ID, removing the booking from the database.
- Following successful cancelation, the affected reservation is removed from the Mytrips display through the setReservations, manipulating the reservations state.



Services

Bus Number	Collection Time	Estimated Travel Time	Pick Up Location	Drop Off Location	Capacity
Search	dd / mm / yyyy		Search	Search	
100	01/03/2024, 11:48:00 am	30	South Bank	Queensland Tennis Center	40
123	23/08/2032, 4:00:00 am	30	South Bank	Queensland Tennis Center	40
		30	South Bank	Queensland Te Center	

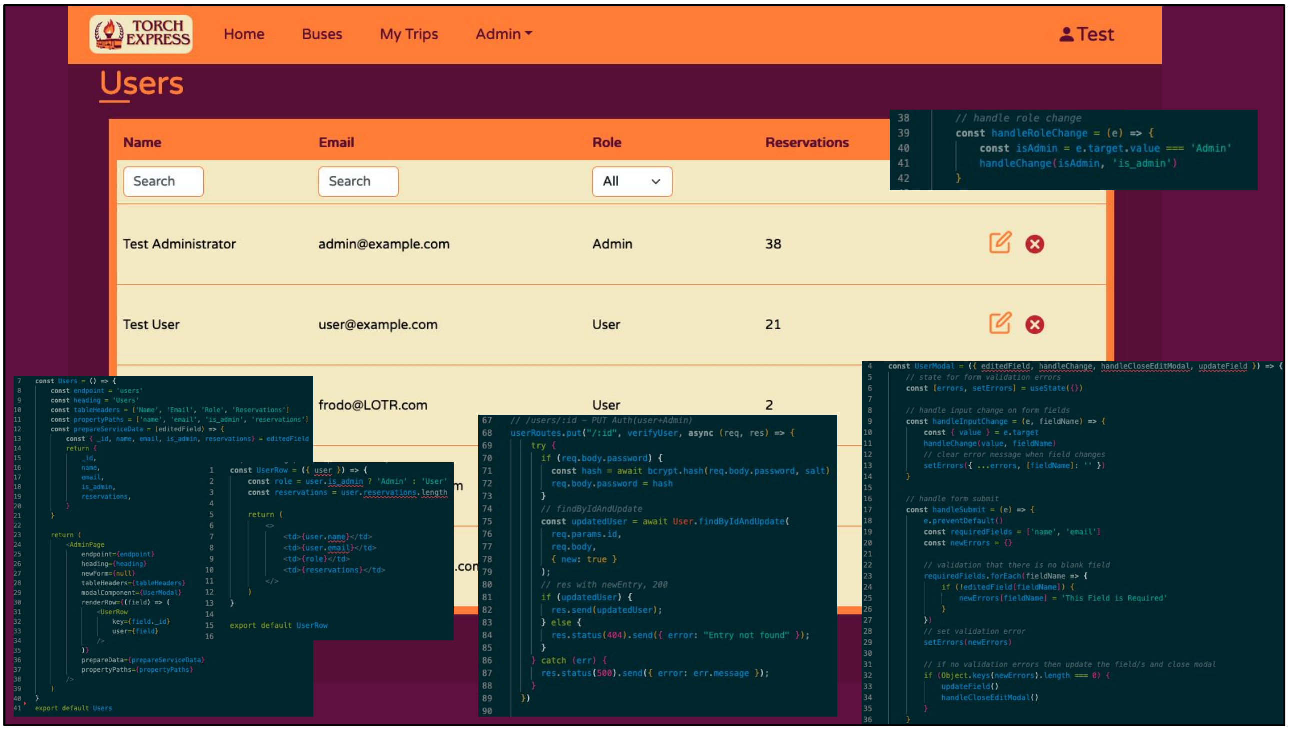
© Torch Express 2024

```javascript
// Admin Page is a reusable component for all the different admin pages
const AdminPage = ({ endpoint, heading, newForm, tableHeaders, modalComponent, renderRow, prepareData, hideEditButton, propertyPaths }) => {
    const [field, setField] = useState([])
    const [showEditModal, setShowEditModal] = useState(false)
    const [editedField, setEditedField] = useState([])
    // generic filter
    const [filter, setFilter] = useState({})
    // filtered data
    const [filterdField, setFilterdField] = useState([])

    // Fetch Data from API
    useEffect(() => {
        fetch(`https://t3a2b-torch-express-api.onrender.com/${endpoint}`,
            {headers : {'Authorization': Cookies.get("accessToken")}}
        )
            .then(res => res.json())
            .then(data => setField(data))
            .catch(error => console.error(`Error fetching ${endpoint}:`, error))
    }, [])

    // Delete
    async function deleteField(id) {
        try {
            await fetch(`https://t3a2b-torch-express-api.onrender.com/${endpoint}/${id}`,
                method: 'Delete',
                headers : {'Authorization': Cookies.get("accessToken")}
            })
            setField(prevField => prevField.filter(item => item._id !== id))
        } catch (error) {
            console.error('Error deleting:', error)
        }
    }
```

```javascript
    // Filter fields on change of filter State
    useEffect(() => {
        setFilterdField(field.filter((row) => Object.entries(filter).every(([key, value]) =>
        value === undefined || JSON.stringify(row[key]).toLowerCase().includes(value.toString().toLowerCase()))
        ))
    }, [field, filter])

    const handleEdit = (field) => {
        setEditedField(field)
        setShowEditModal(true)
    }

    const handleCloseEditModal = () => {
        setShowEditModal(false)
        setEditedField({})
    }
```

```javascript
    const updateField = async () => {
        try {
            const updatedFieldData = prepareData(editedField)

            const response = await fetch(`https://t3a2b-torch-express-api.onrender.com/${endpoint}/${editedField._id}`, {
                method: 'PUT',
                headers: { 'Content-Type': 'application/json', 'Authorization': Cookies.get("accessToken")},
                body: JSON.stringify(updatedFieldData)
            })

            if (!response.ok) {
                throw new Error('Failed to update')
            }

            const updatedField = await response.json()

            setField(prevField => prevField.map(item => (item._id === updatedField._id ? updatedField : item)))
            handleCloseEditModal()

        } catch (error) {
            console.error('Error updating:', error)
        }
    }

    const handleChange = (value, fieldName) => {
        setEditedField(prevState => ({
            ...prevState,
            [fieldName]: value
        }))
    }

```

- The admin pages are only visible to a logged in Admin user. The user is able to navigate to these pages from the drop down menu in the navigation bar
- These pages allow the admin user to create new locations and services, edit locations services and users, and delete locations, services reservations and users.
- The data is fetched from the API using GET requests with the endpoint being passed as a prop into the admin page component.
- The admin pages are created using a reusable admin page component and admin table components - with various props, headers and data being passed through.
- The admin page component uses a series of state hooks and use effect hooks to achieve its functionality.
- All admin pages feature a search bar on the second row of the table which allows user to filter the table data



Users

Name	Email	Role	Reservations
Search	Search	All	
Test Administrator	admin@example.com	Admin	38
Test User	user@example.com	User	21
	frodo@LOTR.com	User	2

```
// handle role change
const handleRoleChange = (e) => {
    const isAdmin = e.target.value === 'Admin'
    handleChange(isAdmin, 'is_admin')
}
```

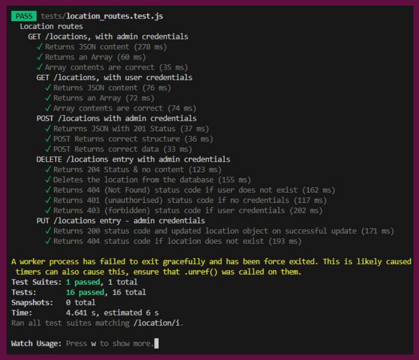
```
const Users = () => {
    const endpoint = 'users'
    const heading = 'Users'
    const tableHeaders = ['Name', 'Email', 'Role', 'Reservations']
    const propertyPaths = ['name', 'email', 'is_admin', 'reservations']
    const prepareServiceData = (editedField) => {
        const { _id, name, email, is_admin, reservations} = editedField
        return {
            _id,
            name,
            email,
            is_admin,
            reservations,
        }
    }

    return (
        <AdminPage
            endpoint={endpoint}
            heading={heading}
            newForm={null}
            tableHeaders={tableHeaders}
            modalComponent={UserModal}
            renderRow={(field) => (
                <UserRow
                    key={field._id}
                    user={field}
                />
            )}
            prepareData={prepareServiceData}
            propertyPaths={propertyPaths}
        />
    )
}
export default Users
```

```
const UserRow = ({ user }) => {
    const role = user.is_admin ? 'Admin' : 'User'
    const reservations = user.reservations.length

    return (
        <>
            <td>{user.name}</td>
            <td>{user.email}</td>
            <td>{role}</td>
            <td>{reservations}</td>
        </>
    )
}

export default UserRow
```

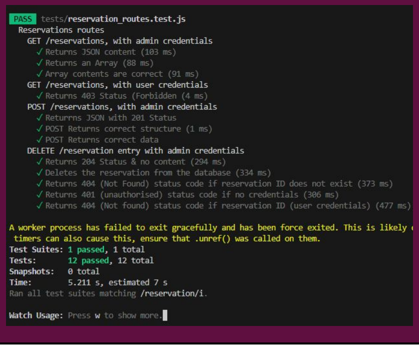
```
// /users/:id - PUT Auth(user+Admin)
userRoutes.put("/:id", verifyUser, async (req, res) => {
    try {
      if (req.body.password) {
        const hash = await bcrypt.hash(req.body.password, salt)
        req.body.password = hash
      }
      // findByIdAndUpdate
      const updatedUser = await User.findByIdAndUpdate(
        req.params.id,
        req.body,
        { new: true }
      );
      // res with newEntry, 200
      if (updatedUser) {
        res.send(updatedUser);
      } else {
        res.status(404).send({ error: "Entry not found" });
      }
    } catch (err) {
      res.status(500).send({ error: err.message });
    }
  })
```

```
const UserModal = ({ editedField, handleChange, handleCloseEditModal, updateField }) => {
    // state for form validation errors
    const [errors, setErrors] = useState({})

    // handle input change on form fields
    const handleInputChange = (e, fieldName) => {
        const { value } = e.target
        handleChange(value, fieldName)
        // clear error message when field changes
        setErrors({ ...errors, [fieldName]: '' })
    }

    // handle form submit
    const handleSubmit = (e) => {
        e.preventDefault()
        const requiredFields = ['name', 'email']
        const newErrors = {}

        // validation that there is no blank field
        requiredFields.forEach(fieldName => {
            if (!editedField[fieldName]) {
                newErrors[fieldName] = 'This Field is Required'
            }
        })
        // set validation error
        setErrors(newErrors)

        // if no validation errors then update the field/s and close modal
        if (Object.keys(newErrors).length === 0) {
            updateField()
            handleCloseEditModal()
        }
    }
```

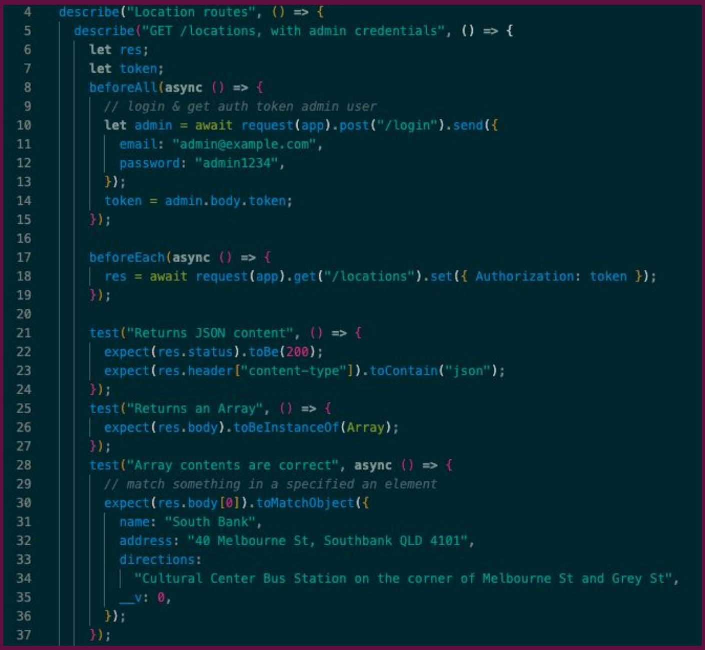
- On the admin page for the user all the users are displayed with the functionality for the admin user to update admin privileges
- All edit modals on the admin pages feature form validation and error messages to ensure the user cannot enter blank fields or fields of the incorrect data type
- On delete a modal will pop prompting confirmation from the user prior delete request being sent to API



```
PASS  tests/location_routes.test.js
  Location routes
    GET /locations, with admin credentials
      ✓ Returns JSON content (278 ms)
      ✓ Returns an Array (60 ms)
      ✓ Array contents are correct (35 ms)
     GET /locations, with user credentials
       ✓ Returns JSON content (76 ms)
       ✓ Returns an Array (72 ms)
       ✓ Array contents are correct (74 ms)
     POST /locations with admin credentials
       ✓ Returrns JSON with 201 Status (37 ms)
       ✓ POST Returns correct structure (36 ms)
       ✓ POST Returns correct data (33 ms)
     DELETE /locations entry with admin credentials
       ✓ Returns 204 Status & no content (123 ms)
       ✓ Deletes the location from the database (155 ms)
       ✓ Returns 404 (Not Found) status code if user does not exist (162 ms)
       ✓ Returns 401 (unauthorised) status code if no credentials (117 ms)
       ✓ Returns 403 (forbidden) status code if user credentials (202 ms)
     PUT /locations entry - admin credentials
       ✓ Returns 200 status code and updated location object on successful update (171 ms)
       ✓ Returns 404 status code if location does not exist (193 ms)

A worker process has failed to exit gracefully and has been force exited. This is likely caused
 timers can also cause this, ensure that .unref() was called on them.
Test Suites: 1 passed, 1 total
Tests:       16 passed, 16 total
Snapshots:   0 total
Time:        4.641 s, estimated 6 s
Ran all test suites matching /location/i.

Watch Usage: Press w to show more.
```



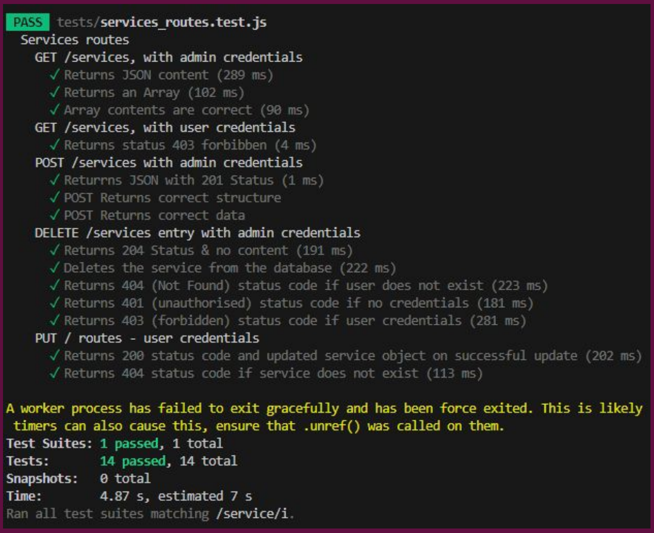
```
PASS  tests/reservation_routes.test.js
  Reservations routes
    GET /reservations, with admin credentials
      ✓ Returns JSON content (103 ms)
      ✓ Returns an Array (88 ms)
      ✓ Array contents are correct (91 ms)
    GET /reservations, with user credentials
      ✓ Returns 403 Status (Forbidden (4 ms)
    POST /reservations, with admin credentials
      ✓ Returrns JSON with 201 Status
      ✓ POST Returns correct structure (1 ms)
      ✓ POST Returns correct data
    DELETE /reservation entry with admin credentials
      ✓ Returns 204 Status & no content (294 ms)
      ✓ Deletes the reservation from the database (334 ms)
      ✓ Returns 404 (Not Found) status code if reservation ID does not exist (373 ms)
      ✓ Returns 401 (unauthorised) status code if no credentials (306 ms)
      ✓ Returns 404 (Not found) status code if reservation ID (user credentials) (477 ms)

A worker process has failed to exit gracefully and has been force exited. This is likely
 timers can also cause this, ensure that .unref() was called on them.
Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        5.211 s, estimated 7 s
Ran all test suites matching /reservation/i.

Watch Usage: Press w to show more.
```



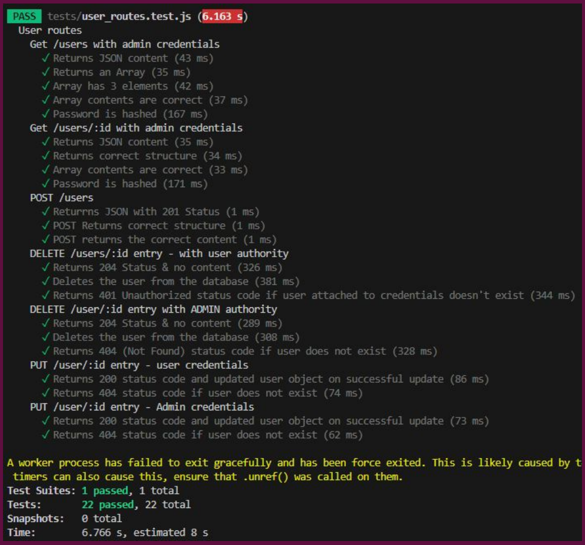
```javascript
describe("Location routes", () => {
  describe("GET /locations, with admin credentials", () => {
    let res;
    let token;
    beforeAll(async () => {
      // login & get auth token admin user
      let admin = await request(app).post("/login").send({
        email: "admin@example.com",
        password: "admin1234",
      });
      token = admin.body.token;
    });

    beforeEach(async () => {
      res = await request(app).get("/locations").set({ Authorization: token });
    });

    test("Returns JSON content", () => {
      expect(res.status).toBe(200);
      expect(res.header["content-type"]).toContain("json");
    });
    test("Returns an Array", () => {
      expect(res.body).toBeInstanceOf(Array);
    });
    test("Array contents are correct", async () => {
      // match something in a specified an element
      expect(res.body[0]).toMatchObject({
        name: "South Bank",
        address: "40 Melbourne St, Southbank QLD 4101",
        directions:
          "Cultural Center Bus Station on the corner of Melbourne St and Grey St",
        __v: 0,
      });
    });
```

- Testing the backend involves ensuring that endpoints respond correctly, handle requests appropriately, and produce the expected output.
- Pictured on the left are passing test suites covering different routes of the API, all four of our API endpoints
- Our API used Jest framework and the Supertest libraries to mimic HTTP requests to the server and testing the responses.
- The tests are set up with describe, to contain unit tests in a suite, categorising tests by routes and function.
- Each unit test is commenced with the "test" function, the Supertest request creates a request to the server, expect is used to assert response data is in the corrected format.
- The example test suite ensures that the "/locations" endpoint behaves as expected. It contains three unit tests.
    a. JSON Content Test: Verifies that the response status is 200 and that the content type header indicates JSON content.
    b. Array Test: Checks if the response body is an instance of an array.
    c. Array Contents Test: Validates the contents of the array by matching the first element against expected values.

API tests passing ctd

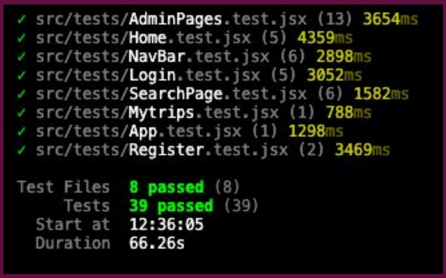


```
PASS  tests/services_routes.test.js
  Services routes
    GET /services, with admin credentials
      ✓ Returns JSON content (289 ms)
      ✓ Returns an Array (102 ms)
      ✓ Array contents are correct (90 ms)
    GET /services, with user credentials
      ✓ Returns status 403 forbibben (4 ms)
    POST /services with admin credentials
      ✓ Returrns JSON with 201 Status (1 ms)
      ✓ POST Returns correct structure
      ✓ POST Returns correct data
    DELETE /services entry with admin credentials
      ✓ Returns 204 Status & no content (191 ms)
      ✓ Deletes the service from the database (222 ms)
      ✓ Returns 404 (Not Found) status code if user does not exist (223 ms)
      ✓ Returns 401 (unauthorised) status code if no credentials (181 ms)
      ✓ Returns 403 (forbidden) status code if user credentials (281 ms)
    PUT / routes - user credentials
      ✓ Returns 200 status code and updated service object on successful update (202 ms)
      ✓ Returns 404 status code if service does not exist (113 ms)

A worker process has failed to exit gracefully and has been force exited. This is likely
 timers can also cause this, ensure that .unref() was called on them.
Test Suites: 1 passed, 1 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        4.87 s, estimated 7 s
Ran all test suites matching /service/i.
```

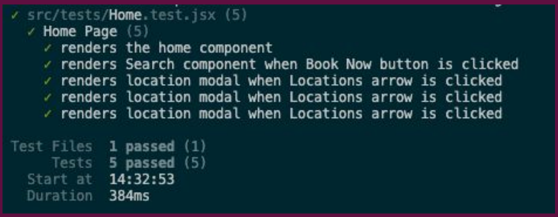


```
PASS  tests/user_routes.test.js (6.163 s)
  User routes
    Get /users with admin credentials
      ✓ Returns JSON content (43 ms)
      ✓ Returns an Array (35 ms)
      ✓ Array has 3 elements (42 ms)
      ✓ Array contents are correct (37 ms)
      ✓ Password is hashed (167 ms)
    Get /users/:id with admin credentials
      ✓ Returns JSON content (35 ms)
      ✓ Returns correct structure (34 ms)
      ✓ Array contents are correct (33 ms)
      ✓ Password is hashed (171 ms)
    POST /users
      ✓ Returrns JSON with 201 Status (1 ms)
      ✓ POST Returns correct structure (1 ms)
      ✓ POST returns the correct content (1 ms)
    DELETE /users/:id entry - with user authority
      ✓ Returns 204 Status & no content (326 ms)
      ✓ Deletes the user from the database (381 ms)
      ✓ Returns 401 Unauthorized status code if user attached to credentials doesn't exist (344 ms)
    DELETE /user/:id entry with ADMIN authority
      ✓ Returns 204 Status & no content (289 ms)
      ✓ Deletes the user from the database (308 ms)
      ✓ Returns 404 (Not Found) status code if user does not exist (328 ms)
    PUT /user/:id entry - user credentials
      ✓ Returns 200 status code and updated user object on successful update (86 ms)
      ✓ Returns 404 status code if user does not exist (74 ms)
    PUT /user/:id entry - Admin credentials
      ✓ Returns 200 status code and updated user object on successful update (73 ms)
      ✓ Returns 404 status code if user does not exist (62 ms)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by t
 timers can also cause this, ensure that .unref() was called on them.
Test Suites: 1 passed, 1 total
Tests:       22 passed, 22 total
Snapshots:   0 total
Time:        6.766 s, estimated 8 s
```

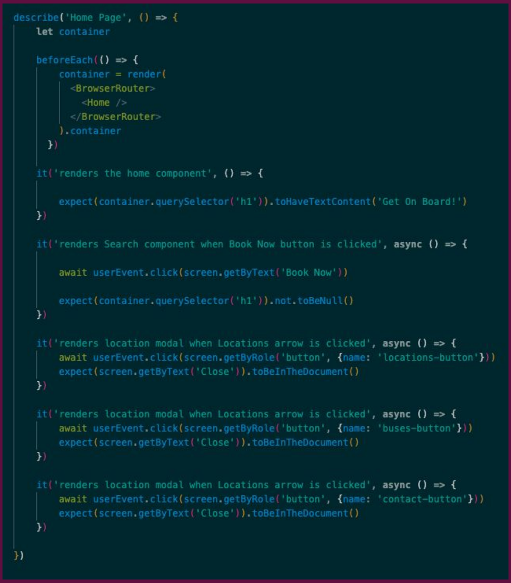
React Testing



```
✓ src/tests/AdminPages.test.jsx (13) 3654ms
✓ src/tests/Home.test.jsx (5) 4359ms
✓ src/tests/NavBar.test.jsx (6) 2898ms
✓ src/tests/Login.test.jsx (5) 3052ms
✓ src/tests/SearchPage.test.jsx (6) 1582ms
✓ src/tests/Mytrips.test.jsx (1) 788ms
✓ src/tests/App.test.jsx (1) 1298ms
✓ src/tests/Register.test.jsx (2) 3469ms

 Test Files  8 passed (8)
      Tests  39 passed (39)
   Start at  12:36:05
   Duration  66.26s
```



```
✓ src/tests/Home.test.jsx (5)
  ✓ Home Page (5)
    ✓ renders the home component
    ✓ renders Search component when Book Now button is clicked
    ✓ renders location modal when Locations arrow is clicked
    ✓ renders location modal when Locations arrow is clicked
    ✓ renders location modal when Locations arrow is clicked

Test Files  1 passed (1)
     Tests  5 passed (5)
  Start at  14:32:53
  Duration  384ms
```



```
describe('Home Page', () => {
    let container

    beforeEach(() => {
        container = render(
          <BrowserRouter>
            <Home />
          </BrowserRouter>
        ).container
      })

    it('renders the home component', () => {

        expect(container.querySelector('h1')).toHaveTextContent('Get On Board!')
    })

    it('renders Search component when Book Now button is clicked', async () => {

        await userEvent.click(screen.getByText('Book Now'))

        expect(container.querySelector('h1')).not.toBeNull()
    })

    it('renders location modal when Locations arrow is clicked', async () => {
        await userEvent.click(screen.getByRole('button', {name: 'locations-button'}))
        expect(screen.getByText('Close')).toBeInTheDocument()
    })

    it('renders location modal when Locations arrow is clicked', async () => {
        await userEvent.click(screen.getByRole('button', {name: 'buses-button'}))
        expect(screen.getByText('Close')).toBeInTheDocument()
    })

    it('renders location modal when Locations arrow is clicked', async () => {
        await userEvent.click(screen.getByRole('button', {name: 'contact-button'}))
        expect(screen.getByText('Close')).toBeInTheDocument()
    })

})
```

- Testing the front end involves verifying that components render correctly, respond to user interactions, and behave as expected.
- This is done in React, using Vitest and the react testing library, to provide testing utilities and Jest, the main testing framework for javascript code
- The beforeEach function rendering the Home component within a BrowserRouter for routing context, that runs before each test case,.
- Describe sets up the test suite for the Home page component.
- Each it block represents an individual test case, visible under when passed, under the HomePage suite in the bottom left figure.
- The userEvent library is used to simulate user interactions like clicks, to trigger assertions made by using expect, to verify specific behaviors or expectations after user interactions occur.
- The example homepage.test.jsx tests if elements on the HomePage render on mount and after user interactions, asserting them to ensure the test passes

Issues encountered:	How you solved them:	Lessons learned:
Initially we faced difficulties in achieving the desired styling for our project and struggled to find a suitable framework that aligned with our team's preference and requirements.	Through dedicated research efforts and open communication, we collectively decided to use React Bootstrap as our styling framework. This helped address our styling concerns and proficiency in utilising the framework	Effective communication and collaborative decision-making is so important in overcoming challenges and finding suitable solutions
When testing in React using Vitest, login and register routes were difficult due to not grasping what exactly is being tested. Using a Mock server seemed illogical to me.	After hours of research into mock data and servers, I ended up testing the actual client server interactions. I accounted for modifications to the database in the testing.	In future products, unit tests should remain successful and be independent in cases of server-side failure.
Not getting nested data when populating data, and not wanting to call to the api again for efficiency sake.	Populate method can go further by populating nested fields within the "busService" object, using path and model, to refer to mongoose schemas	Nested population is an extremely usable method for future mongoose models
Couldn't use the populate method on newly created or updated Mongoose objects to retrieve nested data.	After researching Mongoose documentation and examples, I discovered that I needed to first save the parent object before populating its nested data.	The importance of understanding the timing and sequence of operations when working with Mongoose and how to effectively use the populate method to access nested data.

How did the building of Torch Express go?

- When reflecting it went well, delegation was easily done using Trello and daily standups made it clear to each Team Member what was to be completed.
- Members could call upon others when struggling and having the team dynamic made this assignment achievable.
- Personally I had some struggles with Git, forgetting to pull the latest changes and opening a can of version/rebase hell. Some was solvable, sometimes nothing worked.
- The planning stage really helped in picturing the application and how data interacted with each part of the application.
- Differences in planning stages to what was executed, but that we found natural as having a plan to fall back on made it easier to reference.
- Difficulties arose during the coding stage but through thorough research or finding other solutions, they were managed by our team.