CA2 – Advanced JavaScript

PETER BYRNE | N00153868

Fourth Year
CREATIVE COMPUTING

Introduction

Github Repository - https://github.com/pbtrre95/reactCA2

This React application was built aiming to provide a simple calendar application for event organization. Guests could read events and see what employees were working for each event, while registered users could create, read, update and delete events. The application was built with several different types of user in mind, for example the application could serve:

- As an employee of the business who uses this application, to find which events they are working for to ensure they do not miss their shifts.
- As a manager of the business who wants to see which staff are working for which events to ensure there is a sufficient number of staff working for each event.
- A customer of the business who wants to see their order details, to ensure that there are no mistakes with booking details, number of guests that will be attending etc.
- A manager who wishes to see how many events are scheduled for one date, to ensure they are not overbooked.

Interface Design

The application consists of a single page. The user lands on the homepage which will display a calendar component. The user can click on a certain date within the calendar to display the different events which will occur that given day. There will be some simple details provided like the event ID number and descriptions etc. The user will also be able to see the employees that are working for the event.

From the homepage, the NavBar will hold a number of components allowing the user to register an account, login to their account, create an event or logout if they are presently logged in.

Also depending on whether the user is logged in or not, they will see an option to edit a specific event on the date selected. This will then lead to an edit page, where the user can delete the event too if they are logged in.

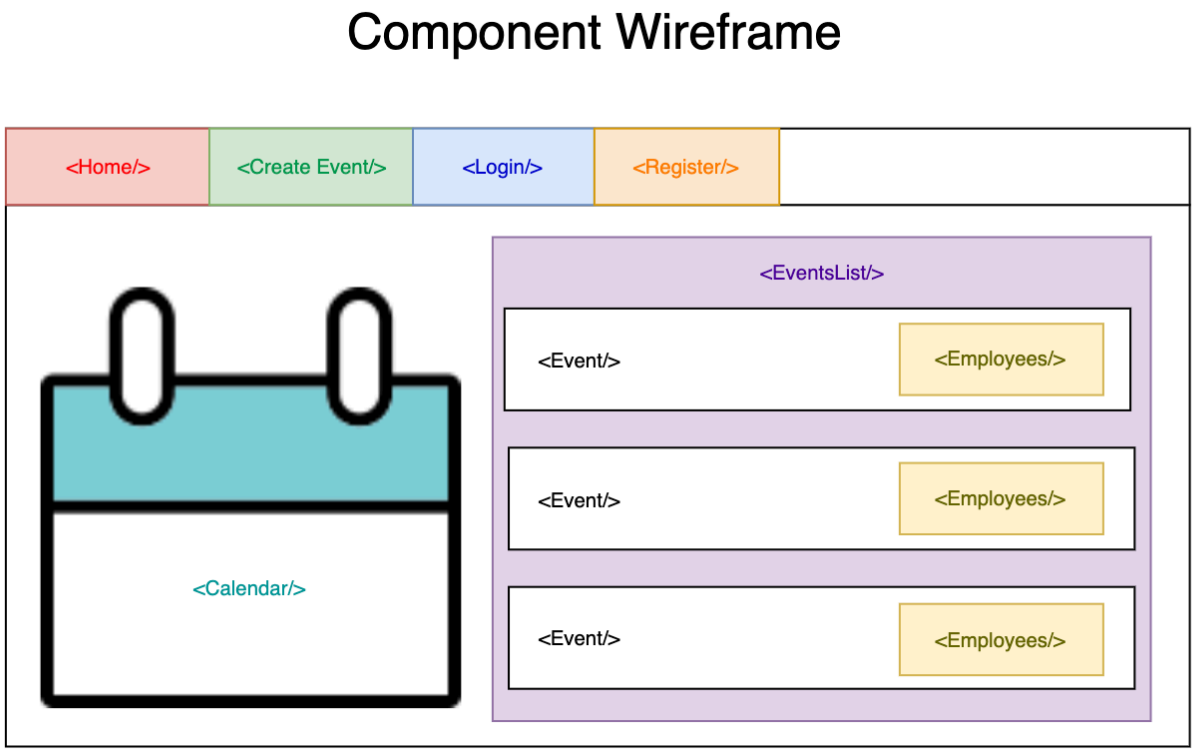


Figure 1 - Component Hierarchy

The above diagram shows the hierarchy of the components rendered.

- The first <Home/> component offers a return to the home screen from the other options.
- Users can create an account with the <Register/> tab.
- Users can create an event at <Create Event/>
- Users can login at <Login/>
- <EventsList/> will display the different events on a given date, they will contain the <Employees/> working for each event
- The <Calendar/> component is rendered on the home screen also

---

Operation and Key Functions

Here is a list of all the possible endpoints built within the application:

- GET /api/home
- GET /api/secret (not used)
- POST /api/register
- POST /api/authenticate
- GET /api/checkToken
- GET /api/logout
- GET /api/events
- GET /api/employees
- GET /api/events:id
- PUT /api/events
- DELETE /api/events
- POST /api/events
- GET /api/events:id/employees

The CRUD operations are working correctly for the Events. Each event may have none or many employees working on it which can be viewed. Some of the more important aspects of the code which allowed the different functionality included the following:

```
<InfiniteCalendar
  onSelect={function(date) {
    today = date;
  }}
  width={1000}
  height={550}
  selected={this.state.today}
  onChange={this.checkLogin()}
/>
```

Figure 2 - Calendar Component

This was the only code required to implement the imported calendar component. Today's date is got using a JavaScript function and sets the calendar's starting date to it. As a date was selected within the calendar, I was required to write the code that would match the dates with the different formatted dates saved in my database. These dates were selected using a HTML input field with a date type so

this needed to be broken down into a relatable format. If the date matched the new formatted date, then the events for that date chosen by the viewer will display.

```
eventList() {
  let day, mon, yr;
  let res = today.toString();
  let months = [];
  day = res.slice(8, 10);
  yr = res.slice(11, 15);
  for (let i = 0; i < months.length; i++) {
    if (res.substring(4, 7) === (months[i][0])) {
      mon = months[i][1];
    }
  }
  let checkDate = yr + '-' + mon + '-' + day;
  return this.state.events.map(function(currentEvent, i) {
    let eventDate = currentEvent.event_date.toString();
    let eventDateShortened = eventDate.slice(0,10);
    if (eventDateShortened === checkDate) {
      return <Event event={currentEvent} key={i} />;
    }
  });
}
```

Figure 3 - EventList component that rendered events with matching date formats to date submitted by the user

I needed to pass the loggedIn state which was held in the App.js file. This was the parent component of the eventsList component. This was done by passing the required props through the React route component as shown below. This was something that was new for me and was key as I used this method in other locations in the application.

5

```
<Route path="/" exact render={(props) => <EventsList {...props} l
```

Figure 4 - Passing Props with React Route

Then, if the user was logged in a button would display allowing them to access an edit page where they can update and delete events.

```
function ifLoggedIn(id) {
  if (loggedIn) {
    return <Link to={'/edit/' + id}><Button className="btn btn-p
  }
  else if (!loggedIn) {
    return '';
  }
}
```

Figure 5 - If User is Logged in Return Edit Functionality

Below is the code required to delete an event.

```
deleteEvent(e) {
  e.preventDefault();

  axios.delete('/api/events/', {
    params: {
      _id: this.props.match.params.id,
    }
  })
    .then(res => console.log(res.data));
  this.props.history.push('/');
}
```

Figure 6 - Delete Axios Request

I changed how the one to many relationship was viewed slightly, showing the one to many relationships on the same page instead of requiring a button to see the links between collections. Below is the event ID being passed to the EmployeeList component. This is then used as a parameter in an Axios request to the database. Below are the Event props being passed to the EmployeeList component and then how that data is displayed within the event component.

```
const Event = props => (
  <tr>
    <td>{props.event._id}</td>
    <td>{changeDate(props.event.event_date)}</td>
    <td>{props.event.event_description}</td>
    <td>{ifLoggedIn(props.event._id)}</td>
    <td>
      <EmployeeList {...props} eventId={props.event._id} />
    </td>
  </tr>
);
```

Figure 7 - Passing Props to the EmployeeList Component

ID	Date	Description	
5ca76e35974287ab97558418	2019-04-15	Party of 20 people, food and drink deals.	All Employees

Figure 8 - Events with Employees Listed

Reflection

I was glad with how this project went as I got some of the major requirements implemented and working congruently in the application. If I was to do it again I would definitely allow myself more time to start the project, so I could work on some of the extra functionalities I kept thinking I could add as I kept coding, I would have liked to have added CRUD operations for the other collections, have different user roles like Guest and Admin and different permissions for each. The project was good in that it gave me a better understanding of some of the more important aspects of React like passing props to components, having more than one collection to work with, creating simple login and register functionality and getting all of these aspects to work together. I even found that writing code for things you would imagine would be relatively easy to implement like a delete Axios request or the different CSS styling requirements for React components, in particular forms, could be trickier than I thought they would be.

References

- The calendar component - https://github.com/claudericc/react-infinite-calendar

Appendices

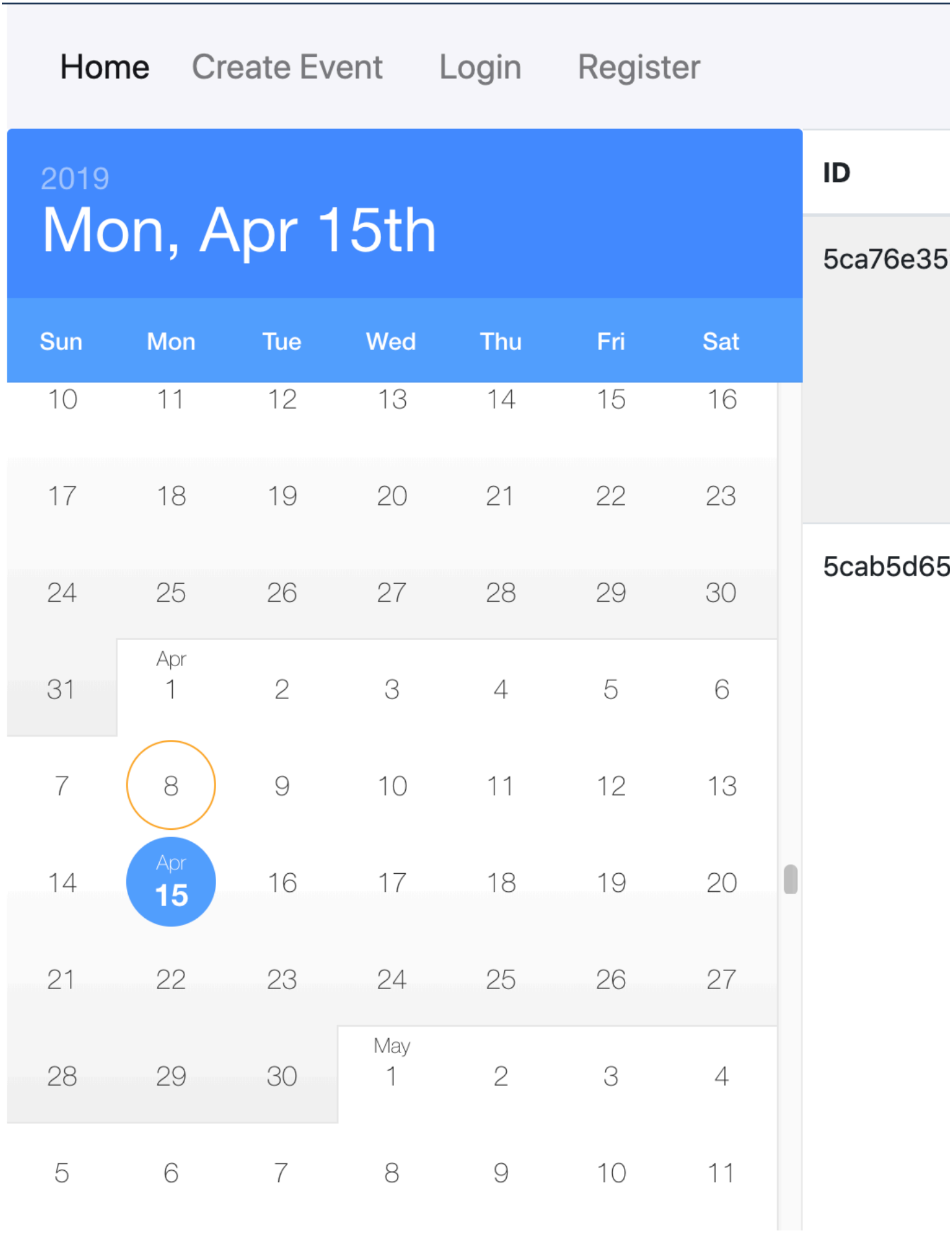

Home
Create Event
Login
Register
2019
Mon, Apr 15th
Sun
Mon
Tue
Wed
Thu
Fri
Sat
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
Apr
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Apr
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
May
1
2
3
4
5
6
7
8
9
10
11
ID
5ca76e35
5cab5d65

Figure 9 – Home Screen

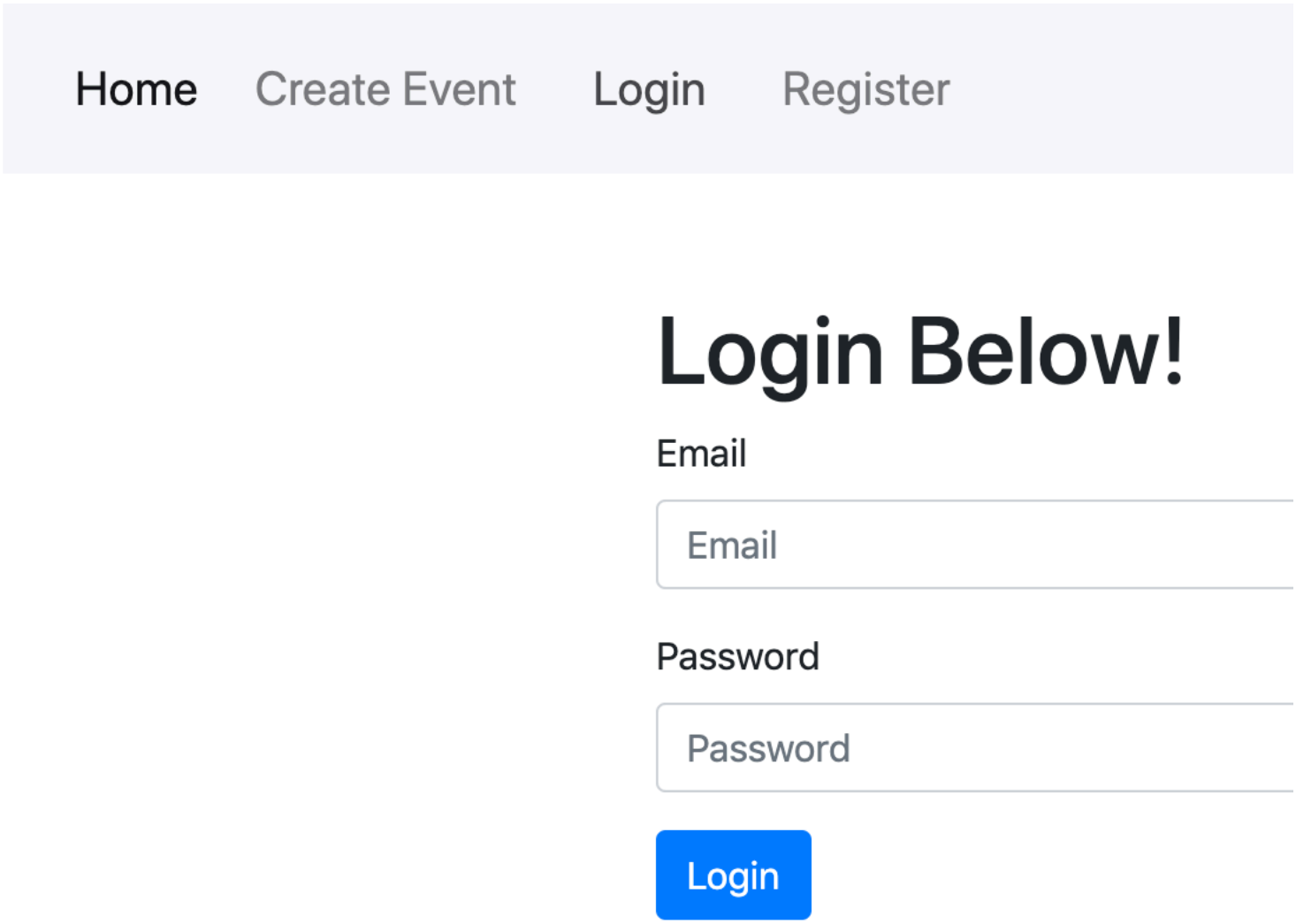
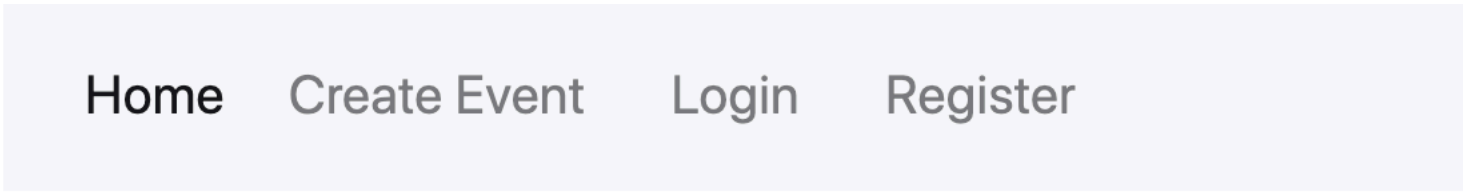

Home
Create Event
Login
Register
Login Below!
Email
Email
Password
Password
Login

Figure 10 - Login Component



Register Below!

First Name

First Name

Last Name

Last Name

Email

Email

Password

Password

Register

Figure 11 - Registration Component

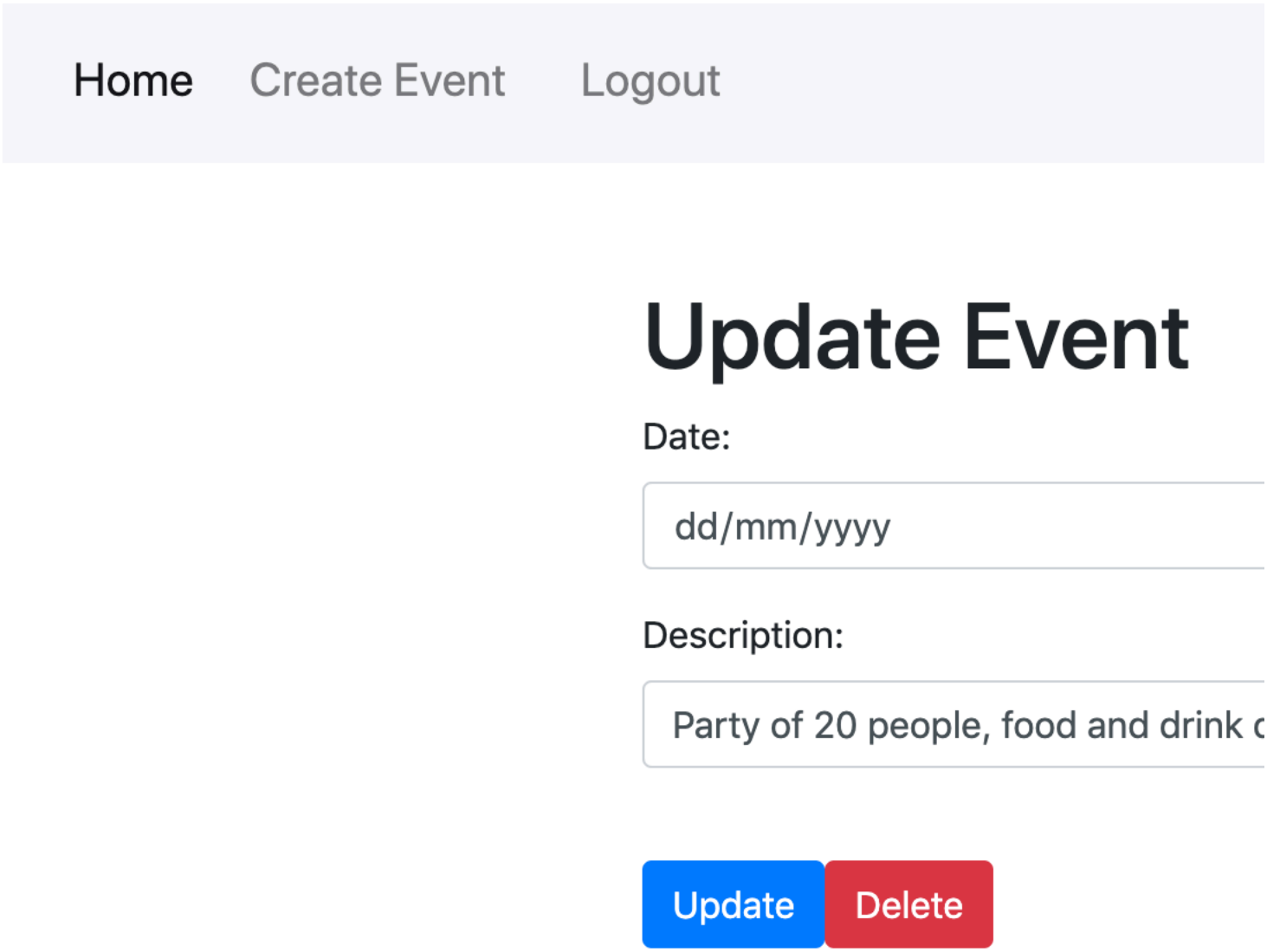


Figure 12 - Update Event

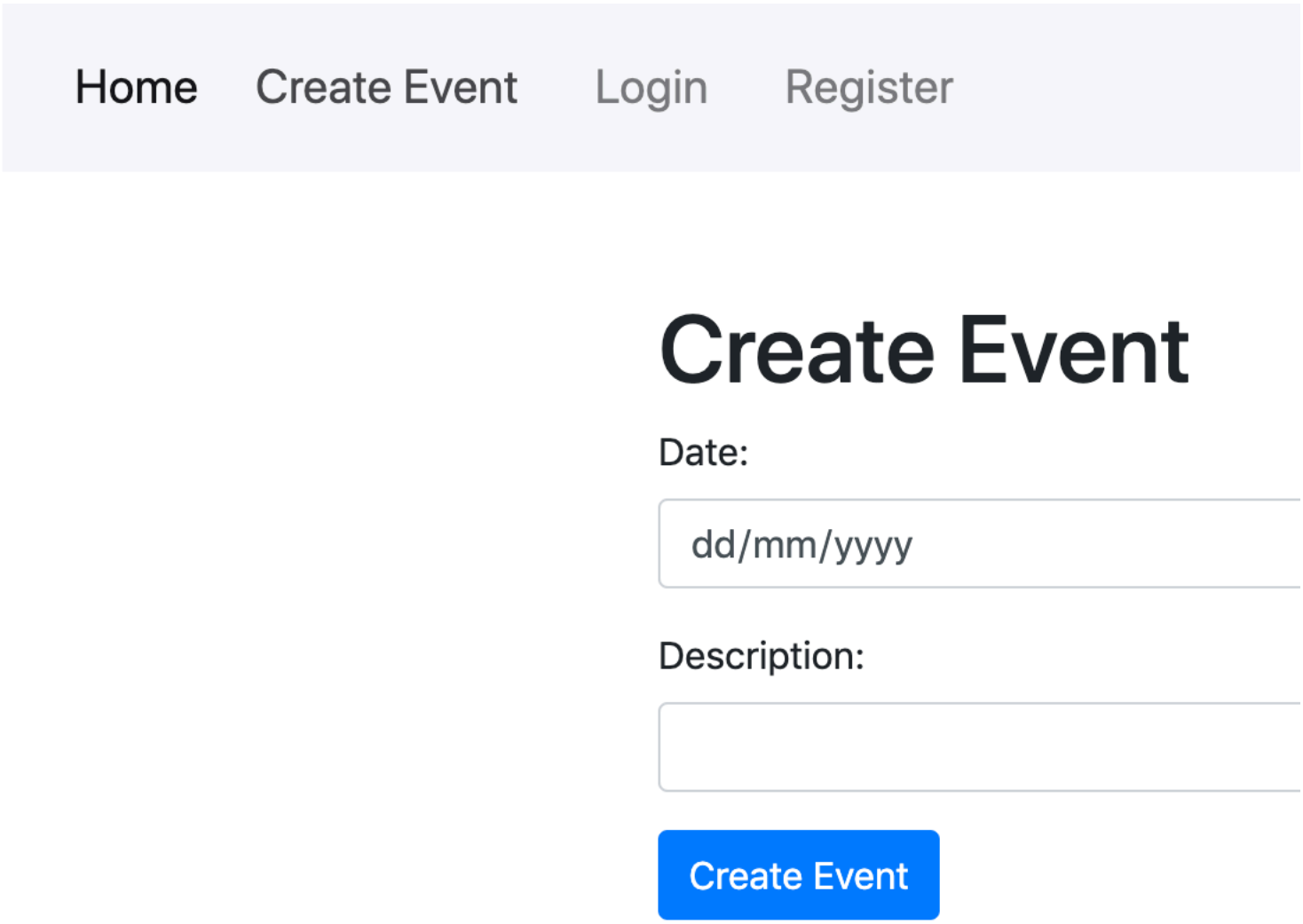


Figure 13 - Create Event