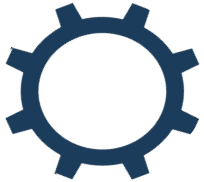
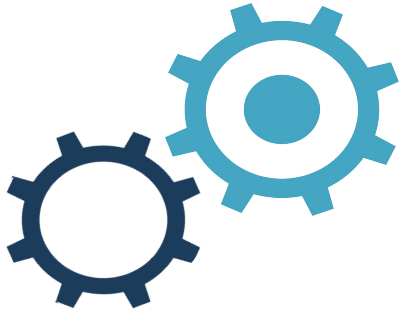


coder ACADEMY

EventScape

Code Explanation and Problems

Hannah Scaife and Steph Elsley



Explanation of Create Events Component(React)

- Allows users to create, save, and publish events with customisable themes via:

State Management and Functionality

- The CreateEvent component uses React's **useState** hook to manage two pieces of state:
    1. **Theme**: Tracks the selected theme for styling.
    2. **EventData**: Stores event information eg title, description, date (when) etc.

```
function CreateEvent() {
  const [theme, setTheme] = useState("default");
  const [eventData, setEventData] = useState({
    eventName: "",
    description: "",
    eventDate: "",
    location: "",
    host: "",
    invited: "",
  });
```

Conditional Statement for Dynamic Theme Switching

- The **useEffect** hook dynamically updates the theme stylesheet(e.g Spiderman.css) based on the selected theme. A **conditional check** ensures the theme stylesheet element exist before modifying it.
- This checks if the link element exists (**if (link)**) and dynamically updates the href attribute of the theme stylesheet whenever the theme state changes.

```
useEffect(() => {
  // Dynamically update the theme
  const link = document.getElementById("theme-stylesheet");
  if (link) {
    link.href = `/themes/${theme}.css`;
  }
}, [theme]);
```

Handling User Input with a Function Call

- The **handleInputChange** function updates the **eventData** state as the user types into input fields. This is achieved using the **onChange** event and dynamically updates state properties. The '(e)' param refers to the element that triggers the event(in this case the input fields)

```
// Handle input field changes (title, description, etc.)
const handleInputChange = (e) => {
  const { name, value } = e.target;
  setEventData({ ...eventData, [name]: value });
};
```

Saving and Publishing the Event

- The **handleSave** and **handlePublish** functions save event data to localStorage and handle publishing the event
- Both functions redirect the user to the saved/published event page

```
// Save the event
const handleSave = () => {
  const savedEvent = { ...eventData, theme };
  localStorage.setItem("savedEvent", JSON.stringify(savedEvent)); // Save to localStorage
  navigate("/saved-event"); // Redirect to saved event page
};
```

Rendering Input Fields with a Loop

- React dynamically renders the input fields and updates the state using reusable logic
- The input fields share a common event handler (**handleInputChange**) that processes updates through a loop-like approach

```
<div className="details">
  <div className="input-group">
    <label>WHEN:</label>
    <input
      type="text"
      placeholder="..."
      name="eventDate"
      value={eventData.when}
      onChange={handleInputChange}
    />
```

- Although not explicitly coded, React's declarative rendering, loops through each state property, via the same onChange function. This approach is similar to looping because the **handleInputChange** dynamically updates the corresponding property in the eventData state based on the **name** attribute.

Problem 1 - State not updating correctly on Search

When the user enters a search, nothing was coming up. Due to issues with how the search input state is managed, the component didn’t trigger a re-render and events didn’t update.

State Management: React uses the useState hook to manage the searchQuery, events, and error states. If the searchQuery state or the events state is not properly updated after the search is triggered, the component can fail to display the correct data.

Re-renders: React re-renders the component when state updates occur, but if the state change isn't correctly detected, it can prevent the events from being updated.

Event Handling: The handleSearch function resets the error state and fetches the events, but if there’s an issue with how the events state is updated (for instance, it’s not updated correctly or cleared before new data arrives), it might not trigger the re-render that displays the updated events list.

Problem 1 - State not updating correctly on Search

Fix: To fix this I updated the logic so that the events state is cleared by resetting the events array before fetching new data and clearing any old data.

I used React documentation and Stack Overflow to figure out the issue.

```
const handleSearchChange = (e) => {
  setSearchQuery(e.target.value);
};

// Handle filter changes
const handleFilterChange = (e) => {
  setFilter(e.target.value);
};

// Fetch and display search results
const handleSearch = async () => {
  if (!searchQuery) {
    setError("Please enter a search query."); // Error for empty search query
    return;
  }


  try {
    setError(null); // Reset error state before a new request

    // Inregrate the filter if applicable
    const filterQuery = filter ? `&category=${encodeURIComponent(filter)}` : '';
    const url = `${import.meta.env.VITE_AUTH_API_URL}/search/public?query=${encodeURIComponent(searchQuery)}${filterQuery}`;

    // Make GET request to backend using fetch
    const response = await fetch(url, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${userJwt.token}`,
      },
    });
```

Explanation of Sign In route (Express.js)

- POST request to the “/signin” Endpoint
- Allows user to sign in via Authentication
- Request body validated using custom Joi validation schema
- Check existing user against database

```
// Sign in existing account
router.post(
    "/signin",
    handleRoute(async (request, response) => {
        const { error } = signinValidationSchema.validate(request.body);
        if (error){
            const errorMessage = error.details.map(detail => detail.message).join(", ").replace(/\"/g, ''
            throw new AppError(errorMessage, 400);
        }

        const { email, password } = request.body;

        const user = await UserModel.findOne({ email: email });
        if (!user) {
            throw new AppError("User not found. Please sign up first.", 404);
        }

        const isPasswordValid = await comparePassword(password, user.password);
        if (!isPasswordValid) {
            console.log("Incorrect password attempt for email:", email);
            throw new AppError("Invalid password.", 401);
        }

        let newJwt = generateJWT(user._id, user.username, user.isAdmin);

        sendSuccessResponse(response, "Sign-in successful.", {
            jwt: newJwt,
            user: {
                id: user._id,
                username: user.username,
                email: user.email,
                isAdmin: user.isAdmin
            }
        });
    })
);
```

Sign In Validation Schema

- Uses Joi validation Library
- Allows for robust validation and error handling

```
const signinValidationSchema = Joi.object({
    email: Joi.string()
        .email()
        .required()
        .messages({
            "string.empty": "Email is required.",
            "string.email": "Please provide a valid email address."
        }),
    password: Joi.string()
        .min(8)
        .pattern(new RegExp('^(?=.*[A-Za-z])(?=.*\\d)(?=.*[@$!%*#?&])[A-Za-z\\d@$!%*#?&]{8,}$'))
        .required()
        .messages({
            "string.empty": "Password is required.",
            "string.min": "Password must be at least 8 characters long.",
            "string.pattern.base": "Password must include at least one letter, one number, and one
            special character."
        }),
});
```

ValidateUserAuth Middleware

- Middleware with dual functionality
- Used for authorthorisation & authentication

```
// Middleware to authenticate user / ensure they are logged in

async function validateUserAuth(request, response, next) {
    const providedToken = request.headers.authorization && request.headers.authorization.split(' ')[1];

    if (!providedToken) {
        return response.status(403).json({
            message: "Please sign in to view this content."
        });
    }

    try {
        const decodedData = decodeJWT(providedToken, process.env.JWT_SECRET_KEY);
        console.log("Decoded data: ", decodedData);

        if (decodedData && decodedData.userId) {
            request.authUserData = decodedData;
            return next();
        } else {
            return response.status(403).json({
                message: "Please sign in to view this content.",
            });
        }
    } catch (error) {
        if (error.name === 'TokenExpiredError') {
            return response.status(401).json({
                message: "Your session has expired. Please log in again."
            });
        }

        return response.status(403).json({
            message: "Invalid token. Please sign in to view this content."
        });
    }
}
```

Issues: CORS Policy

- Preflight request error
- CORS header ‘Access-Control-Allow-Origin’ Header missing
- Fixes: methods, credentials, Preflight request route handler

```
let corsOptions = {
    origin: function (origin, callback) {
        const validOrigins = [
            /^http:\/\/localhost:\d+$/,
            /^http:\/\/127\.0\.0\.1:\d+$/,
            /^https:\/\/eventscape\d*\.netlify\.app$/,
            "https://eventscape1.netlify.app"
        ];

        // Check if the origin is in the allowed list or if it's a non-browser request
        if (validOrigins.includes(origin)){
            // Allow request
            callback(null, true);
        } else {
            console.warn(`CORS blocked request from origin: ${origin}`)
            const err = new Error("Not allowed by CORS");
            err.status = 403;
            callback(err, false);
        }
    },
    methods: "GET,PATCH,POST,DELETE,OPTIONS",
    credentials: true,
    optionsSuccessStatus: 200
};

app.use(cors(corsOptions));

// Pre-flight handler - must be above routes
app.options("*", cors(corsOptions));
```

RSVP Controller - ValidatorError

```
Error updating RSVP response:  Error: RSVP validation failed: status: Path `status` is required.
    at ValidationError.inspect (/Users/hannahscaife/Documents/GitHub/EventScape/node_modules/mongoose/lib/error/validation.js:52:26)
    at formatValue (node:internal/util/inspect:806:19)
    at inspect (node:internal/util/inspect:365:10)
    at formatWithOptionsInternal (node:internal/util/inspect:2304:40)
    at formatWithOptions (node:internal/util/inspect:2166:10)
    at console.value (node:internal/console/constructor:347:14)
    at console.error (node:internal/console/constructor:392:61)
    at updateRSVP (/Users/hannahscaife/Documents/GitHub/EventScape/src/utils/crud/RSVPCrud.js:72:17)
    at process.processTicksAndRejections (node:internal/process/task_queues:105:5)
    at async /Users/hannahscaife/Documents/GitHub/EventScape/src/controllers/RSVPController.js:39:33 {
  errors: {
    status: ValidatorError: Path `status` is required.
        at validate (/Users/hannahscaife/Documents/GitHub/EventScape/node_modules/mongoose/lib/schemaType.js:1404:13)
        at SchemaType.doValidate (/Users/hannahscaife/Documents/GitHub/EventScape/node_modules/mongoose/lib/schemaType.js:1388:7)
        at /Users/hannahscaife/Documents/GitHub/EventScape/node_modules/mongoose/lib/document.js:3080:18
        at process.processTicksAndRejections (node:internal/process/task_queues:85:11) {
      properties: [Object],
      kind: 'required',
```

```
An error occurred:  AppError: Error updating RSVP status, please try again later.
    at updateRSVP (/Users/hannahscaife/Documents/GitHub/EventScape/src/utils/crud/RSVPCrud.js:78:15)
    at process.processTicksAndRejections (node:internal/process/task_queues:105:5)
    at async /Users/hannahscaife/Documents/GitHub/EventScape/src/controllers/RSVPController.js:39:33
    at async /Users/hannahscaife/Documents/GitHub/EventScape/src/middleware/routerMiddleware.js:9:13 {
  statusCode: 500,
  isOperational: true
}
```

```
async function handleRSVPStatus(event, userId, status){
    if (!["yes", "maybe", "no"].includes(status)){
        throw new AppError("Invalid response type. Status must be 'yes', 'maybe' or 'no'.", 400);
    }

    if (event.host.toString() === userId.toString()){
        throw new AppError("The host cannot RSVP to their own event.", 403);
    }

    if (!event.isPublic && !event.invited.includes(userId)){
        throw new AppError("This is a private event.", 403);
    }

    if (event.attendees.includes(userId) && status === "yes") {
        throw new AppError("You have already RSVP'd as 'yes' to this event.", 400);
    }

    if (event.maybe_attending.includes(userId) && status === "maybe") {
        throw new AppError("You have already RSVP'd as 'maybe' to this event.", 400);
    }

    if (event.not_attending.includes(userId) && status === "no") {
        throw new AppError("You have already RSVP'd as 'no' to this event.", 400);
    }

    switch (status) {
        case "yes":
            if (!event.attendees.includes(userId)) {
                event.attendees.push(userId);
            }
            event.not_attending = event.not_attending.filter(user => user.toString() !== userId.toString());
                event.maybe_attending = event.maybe_attending.filter(user => user.toString() !== userId.toString());
            break;
        case "no":
            if (!event.not_attending.includes(userId)) {
                event.not_attending.push(userId);
            }
            event.attendees = event.attendees.filter(user => user.toString() !== userId.toString());
```

```
async function updateRSVP(query, updatedData){
    try {
        const { eventId, userId, status } = updatedData;
        console.log("Received status in updateRSVP:", updatedData.status);
        console.log("Status received in request:", status);
```

```
const rsvp = await RSVPModel.findOneAndUpdate(
    { eventId: eventId, userId: userId },
    { $set: updatedData },
    { new: true, runValidators: true }
  ).catch(error => {
    console.log("Mongoose update error:", error);
    throw new Error('Error updating RSVP status');
  });
```

```
const newRSVP = await RSVPModel.create(data);
```