AMIS 4630 — Foundations of Business Information Systems

Week 6 Workshop: React Components & .NET Controllers Practice

Friday, February 20 | 55 minutes | Collaboration: O (Allowed)

What We're Doing Today

You're going to clone a partially-built event ticketing application and complete three exercises. This is practice — not your Buckeye Marketplace project.

Why a different domain? Because I want to make sure you understand the concepts, not just copy your existing code. The patterns in an event ticketing app are identical to what you'll use in M3.

By the end of the workshop, you should have:

- A working EventCard component that receives and displays event data via props
- A TicketCounter component that uses useState to track a running count
- Two completed API endpoints in EventsController.cs

Setup (Do This First)

Run these commands in your terminal:

```
git clone https://github.com/amis-4630/amis4630-workshop-w6.git
cd amis4630-workshop-w6/frontend
npm install
npm run dev
```

Open **`http://localhost:5173`** — you should see 'OSU Event Finder' with a list of event titles. The titles are rendered from hardcoded data. Your job is to replace the plain text with actual components.

Stuck on setup? Raise your hand now — don't spend the workshop troubleshooting environment issues.

Exercise 1 — EventCard Component (Props)

Estimated time: 12 minutes

What you're building

A component that receives a single event object as a prop and displays it as a card. Open `src/components/EventCard.tsx` — it has a stub waiting for you.

Requirements

- Create and event type
- Accept an event (the full event object)
- Display: title, date, venue, price, category
- The price should be displayed as a dollar amount. If the price is 0, show 'Free' instead
- Apply at least minimal styling so it looks like a card, not a wall of text

Wire it up

Once EventCard is built, open EventList.tsx and replace the placeholder <div> with your EventCard. Don't forget:

- Import EventCard at the top of EventList.tsx
- Pass the event object as a prop: <EventCard event={event} />
- Keep the key prop on the outer element

Thinking prompt: What's the difference between props.event.title and destructuring { event } in the function signature? Both work — which feels cleaner to you and why?

Check your work

- The browser shows 6 event cards
- Each card displays all required fields
- Free events show 'Free', paid events show a dollar amount

Exercise 2 — TicketCounter Component (useState)

Estimated time: 12 minutes

What you're building

A component that lets a user select how many tickets they want for an event. Open src/components/TicketCounter.tsx.

Requirements

- Accept two props: eventTitle (string) and maxTickets (number)
- Use useState to track the selected ticket count (start at 0)
- Show a − button, the current count, and a + button
- The − button should not let the count go below 0
- The + button should not let the count go above maxTickets
- Show a message when count > 0: e.g., '2 tickets selected for Buckeye Basketball.'

Add it to the app

In App.tsx, add a TicketCounter below your EventList. Pass in any event title and maxTickets={4}. You're just testing it works — it doesn't need to connect to a real event yet.

Thinking prompt: Why does React re-render the component when you call setCount(), but not when you write count = count + 1 directly? What's actually happening?

Check your work

- Clicking + increases the count; clicking − decreases it
- Can't go below 0 or above maxTickets
- Message appears when tickets > 0

Exercise 3 — EventsController (.NET API)

Estimated time: 12 minutes

What you're building

Two API endpoints in the partially-built EventsController.cs. The class, data, and routing attributes are already there. You're adding the action methods.

Switch to the API project

```
# In a new terminal tab
cd amis4630-workshop-w6/api
dotnet run
```

Once it's running, open `https://localhost:5000/swagger` to see the API explorer.

Endpoint 1: GET /api/events

- Returns all events as a JSON array
- HTTP 200 status
- Use return Ok(_events)

Endpoint 2: GET /api/events/{id}

- Accepts an integer id in the route
- Finds the matching event in _events
- Returns HTTP 200 with the event if found
- Returns HTTP 404 if no event with that id exists

Thinking prompt: What attribute goes on the method for a route with a parameter? How does .NET know that {id} in the route maps to the int id in your method signature?

Update the api.http test file

- Specify the right base route
- Tests for both endpoints

Check your work

- GET /api/events returns a JSON array of 6 events in Swagger
- GET /api/events/1 returns the basketball game
- GET /api/events/99 returns a 404 response

Extension (If You Finish Early)

Pick one:

- **Filter by category:** Add a category filter button in React (e.g., 'Sports only'). Use useState to track the selected category and filter the events array before passing to EventList.
- **GET /api/events?category=Sports:** Add an optional query parameter to your GET /api/events endpoint that filters by category. Test it in Swagger.
- **Styling pass:** Make your EventCard look like something you'd actually ship. Grid layout, hover state, category badge with color.

Before You Leave

Submit your work — it counts toward participation.

Before the end of class, be ready to answer if called on:

- Where does the event data come from in Exercise 1? Trace it from App.tsx to EventCard.
- Why does updating the state trigger a re-render, but updating a regular variable doesn't?
- What does the [controller] token do in the .NET route attribute?

Looking Ahead

Week 7 is where these two sides connect. Your React app will call your .NET API using fetch(), and the data flowing through your components will come from a real HTTP response — not a hardcoded array.

That's the integration step. Everything you practiced today is the prerequisite.