Session 4 Instructor Guide: Data-Driven Design

Learning Outcomes

By the end of Session 4, students will be able to:

1. **Explain metadata and data-driven architecture** and why configuration files separate data from code
2. **Describe the benefits** of configuration files for maintainability, collaboration, and testing
3. **Create JavaScript arrays** using bracket syntax and apply zero-based indexing
4. **Build JavaScript objects** using curly brace syntax with key-value properties
5. **Apply different data types** including strings, numbers, booleans, and objects in object properties
6. **Combine data structures** through nesting arrays of objects and objects with object properties
7. **Use React Fragments** to return multiple elements without extra DOM nodes
8. **Configure zone metadata** by updating the ZONES array with custom game content
9. **Position UI elements** using coordinate systems and real-time feedback tools
10. **Explore OpenTrivia DB documentation** to identify category IDs for zone configuration
11. **Test configuration changes** using React DevTools to manipulate state and observe effects
12. **Demonstrate a working zone configuration** with three themed zones and correctly positioned labels

Instruction

Instructor introduces key concepts students need to succeed:

1. **Data-Driven Game Design** - Introduce the concept of separating game content from game logic using configuration files
2. **JavaScript Data Structures** - Define arrays and objects as the building blocks for complex data — the tools you'll use to describe your game zones
3. **Metadata and Configuration** - Explain how zone metadata describes game content without being the content itself

4. **OpenTrivia DB Integration** - Explore the external API that provides trivia questions and categories — you’ll choose category IDs to match your zone themes
5. **React Fragments** - Solve the single root element requirement when returning multiple components
6. **Coordinate Positioning** - Use the CoordinateDisplay tool to find exact (x, y) positions for your zone labels — and see them update in real time
7. **Nested Data Structures** - Show how arrays of objects with object properties represent complex game data
8. **Development Patterns** - Emphasize configuration-driven design as a widely-used best practice
9. **State Management Testing** - Use React DevTools to simulate zone completion and test game behavior — without needing to play through each level
10. **Zone Theme Planning** - Guide students through cohesive theme creation that matches visual and content elements
11. **Let’s Configure!** - Launch the coding mission: design zones, update metadata, position labels, and test your game world

---

Slide Deck Outline

Slide 1: Data-Driven Design 🏗️

- **Title:** “Session 4: Data-Driven Design — Configuring Game Zones”
- **Session 3 Recap:** “Last time: You managed game flow with shared state, implemented local state for modals, and gained React DevTools expertise”
- **Hook:** “Your game has structure — now you’re the architect of its content!”
- **Today’s Mission:**
  - **Design** cohesive zone themes that create immersive experiences
  - **Explore** JavaScript data structures (arrays and objects)
  - **Configure** game metadata using data-driven architecture
  - **Position** UI elements with coordinate systems
  - **Test** configurations using React DevTools state manipulation
- **Visual:** Game map with three distinct themed zones
- **Connection:** “You’ve built the navigation system — now you’ll populate it with rich, configurable content!”

Slide 2: Data-Driven Architecture - The Modern Approach 📋

- **Title:** “Separating Data from Code”
- **The Problem:** Hardcoded game content scattered throughout components
- **The Solution:** Configuration files that define game content separately
- **Benefits:**
  - **Maintainability** - Change content without touching component code
  - **Scalability** - Add new zones by updating data, not components
  - **Collaboration** - Designers can modify content without coding
  - **Testing** - Easy to test different configurations
- **Real Example:** `zones.js` defines all zone properties in one place
- **Real-World Context:** “Streaming apps and gaming companies use this pattern for content management”
- **Student Preview:** “Your zones.js file is the DNA of your game experience”

Slide 3: Metadata - Data About Data 🏷️

- **Title:** “Understanding Metadata in Application Development”
- **Definition:** “Information that describes other information”
- **Game Context:** Zone metadata describes how to get and display trivia questions
- **Examples:**
  - **Content metadata:** Category ID, difficulty, question count
  - **Display metadata:** Zone name, subtitle, visual styling
  - **Position metadata:** Map coordinates, font size, color
- **Key Insight:** “Metadata isn’t the trivia questions — it’s the instructions for getting and showing them”
- **Analogy:** “Like a recipe card that tells you what ingredients to buy and how to cook them”
- **Student Connection:** “You’ll design metadata that defines your entire game experience”

Slide 4: JavaScript Data Structures - Arrays and Objects 🏗️

- **Title:** “Arrays and Objects — Your Tools for Game Configuration”
- **Visual:** Side-by-side comparison with syntax highlighting

Arrays `[]` - Ordered Lists:

```
const zones = [zone0, zone1, zone2]; // Three zones in order
zones[0] // First zone (zero-indexed)
```

Objects `{}` - Key-Value Collections:

```
const zone = {
  name: "Forest of Knowledge",
  difficulty: "easy",
  questionCount: 4
};
zone.name // Access property with dot notation
```

- **Complementary Tools:**
  - **Arrays** - Perfect for ordered collections (your three zones)
  - **Objects** - Ideal for structured data with named properties (zone details)
  - **Together** - Arrays of objects combine both strengths
- **Nesting Power:** Arrays of objects, objects with object properties
- **Student Preview:** “Your ZONES array contains three zone objects, each with multiple properties”
- 🎯 **Instructor Demo (90s):** Open `zones.js`, edit `ZONES[0].name` from current value to “My Custom Zone”, and show UI update in browser

Slide 5: Data Types in JavaScript 📊

- **Title:** “Working with Different Types of Information”
- **Visual:** Color-coded examples showing different data types

Common Data Types:

Type	Example	Usage in Zones
String	`"Forest of Knowledge"`	Names, subtitles, difficulty levels
Number	`18`, `4`, `225`	Category IDs, question counts, coordinates

Boolean	`true`, `false`	Completion status, visibility flags
Object	`{ x: 225, y: 140 }`	Map label styling, nested configuration
Array	`[zone0, zone1, zone2]`	The ZONES collection itself

- **Key Rule:** “Strings need quotes, numbers don’t, objects use `{}`, arrays use `[]`”
- **Student Connection:** “You’ll use all these types in your zone configuration”

Slide 6: React Fragments - Clean Component Returns 🧩

- **Title:** “React Fragments — Snap Pieces Together Without Extra Wrappers”
- **The Problem:** React components must return a single root element
- **Bad Solution:** Wrapper divs that clutter your HTML
- **Good Solution:** React Fragments `<>...</>`

Before (Messy):

```
return (
  <div> {/* Unnecessary wrapper */}
    <GameMap />
    <HUD />
    <CoordinateDisplay />
  </div>
);
```

After (Clean):

```
return (
  <>
    <GameMap />
    <HUD />
    <CoordinateDisplay />
  </>
);
```

- **Benefits:** Clean HTML output, no styling conflicts from wrapper divs

- **Alternative Syntax:** `<React.Fragment>...</React.Fragment>` (same result)
- **Student Application:** "You'll use fragments to add HUD and CoordinateDisplay to your game screen"

Slide 7: OpenTrivia DB - Your Question Source 🌐

- **Title:** "Exploring Real Trivia Data"
- **Live Demo:** Visit https://opentdb.com/api_category.php
- **What You'll See:** JSON data showing all available trivia categories
- **Key Information:**
  - **Category IDs** - Numbers that identify question types
  - **Category Names** - Descriptive labels for each topic
  - **Variety** - Science, history, entertainment, sports, and more
- **Design Process:**
  1. **Browse categories** - Find topics that match your zone themes
  2. **Note category IDs** - Numbers you'll use in configuration
  3. **Plan cohesive themes** - You choose the trivia categories that shape your zone themes
- **Real-World Context:** "Real apps integrate with external APIs for dynamic content"
- **Student Mission:** "Choose categories that create immersive, themed experiences"
- 🎯 **Student Activity (2-3 min):** Have students browse the API categories and write down 3 category IDs that match their planned zone themes (Forest/easy, Desert/medium, Ice Castle/hard)

Slide 8: Coordinate Positioning - Precise UI Placement 📍

- **Title:** "Place Your Labels Like a Pro — Using Coordinates"
- **Coordinate System Basics:**
  - **Origin (0,0)** - Top-left corner (same as browser window and game map canvas)
  - **X-axis** - Horizontal position (left to right)
  - **Y-axis** - Vertical position (top to bottom)
- **CoordinateDisplay Tool:**
  - **Real-time feedback** - Shows mouse position as you move
  - **Precise placement** - Find exact coordinates for zone labels

- **Visual testing** - See immediately where elements will appear

- **Real-World Usage:** “Game developers and UI designers use coordinate systems for precise layouts”
- **Student Workflow:**
  1. Move mouse around game map
  2. Note coordinates for good label positions
  3. Update zone configuration with chosen coordinates
  4. Test and adjust as needed
- 🎯 **Instructor Demo (90s):** Navigate to game screen, move mouse over map while showing CoordinateDisplay, record coordinates (e.g., x: 300, y: 200), paste into `ZONES[0].mapLabel.x` and `ZONES[0].mapLabel.y`, refresh browser to show label repositioning

Slide 9: Nested Data Structures - Complex Information Modeling 🪆

- **Title:** “Representing Real-World Complexity in Code”
- **Visual:** Nested structure diagram showing ZONES array breakdown

```
ZONES = [                     // Array of zones
  {                          // Zone object
    name: "Forest",          // String property
    categoryId: 18,          // Number property
    mapLabel: {              // Object property
      x: 225,                // Number in nested object
      y: 140,                // Number in nested object
      color: "#333"          // String in nested object
    }
  }
  // ... more zone objects
]
```

- **Why Nesting Matters:**
  - **Organization** - Group related properties together
  - **Flexibility** - Different zones can have different styling
  - **Maintainability** - Clear structure makes updates easier
- **Access Patterns:** `ZONES[0].mapLabel.x` - Drill down through the structure

- **Student Connection:** “Your zone configuration uses arrays of objects with nested object properties”

Slide 10: Zone Theme Planning - Cohesive Game Theming 🎨

- **Title:** “Planning Cohesive Zone Experiences”
- **Design Principles:**
  - **Visual coherence** - Match trivia categories to zone environments
  - **Difficulty progression** - Start easy, increase challenge
  - **Player engagement** - Choose interesting, varied topics
- **Theme Examples:**
  - **Forest Zone** - Nature, animals, science (easy difficulty)
  - **Desert Zone** - History, geography, mythology (medium difficulty)
  - **Ice Castle Zone** - Entertainment, sports, art (hard difficulty)
- **Planning Workflow:**
  1. **Brainstorm themes** - What matches your visual environments?
  2. **Research categories** - Browse OpenTrivia DB for options
  3. **Plan progression** - Easy to hard difficulty curve
  4. **Design cohesively** - Names and subtitles that fit themes
- **Key Insight:** “Game designers spend significant time on thematic coherence”

Slide 11: Design Your Game World! 🚀

- **Today’s Coding Mission:**
  1. **Add HUD components** - Import and use React Fragments
  2. **Explore data structures** - Examine the ZONES array structure
  3. **Design zone themes** - Plan cohesive experiences with category research
  4. **Configure metadata** - Update zone objects with custom content
  5. **Position labels** - Use CoordinateDisplay for precise placement
  6. **Test with DevTools** - Manipulate state to verify configurations
- **Success Criteria:**
  - Three complete zone configurations with unique themes
  - Properly positioned zone labels on the game map

    - HUD displaying current zone information
- **Development Workflow:** “Plan first, implement systematically, test thoroughly”

[HANDS-ON WORK HAPPENS HERE]

Slide 12: React DevTools - Configuration Testing 🔍

- **Title:** “Testing Game Scenarios Without Playing Through”
- **State Manipulation Workflow:**
    1. **Find GameProvider** - Locate in Components tab
    2. **Examine zoneProgress** - See array of zone completion states
    3. **Modify completion status** - Change `completed: false` to `true`
    4. **Observe cascading effects** - Watch activeZone and currentZone update
    5. **Check UI updates** - See HUD reflect new game state
- **Key Benefits:**
    - **Rapid testing** - No need to play through entire game
    - **Edge case exploration** - Test unusual game states
    - **Debug assistance** - Understand state relationships
- **Student Empowerment:** “You can test any game scenario instantly”

Slide 13: What’s Next - Connecting to External APIs 🌐

- **Title:** “Preview of Session 5”
- **Today’s Achievement:** “You designed game content using data-driven architecture”
- **Next Challenge:** “Connect to real trivia APIs and handle dynamic data”
- **Concepts Coming:**
    - **API integration** - Fetch questions from OpenTrivia DB
    - **Async JavaScript** - Handle network requests and promises
    - **Data transformation** - Process API responses for your game
- **Motivation:** “Your zones will load real trivia questions from the internet!”
- **Visual:** Preview of API data flow from OpenTrivia DB to game components