Session 4 — Configuring Game Zones

Data-Driven Architecture 🗺️

You're about to design the heart of your trivia game — the zone configuration that defines your entire game experience! This guide walks you through creating cohesive zone themes, understanding JavaScript data structures, and building the metadata that powers your adventure. Ready to architect your game world? Let's go!

Table of Contents

- Access Your Codespace
- Adding the HUD and Coordinate Display
- Understanding Metadata and Configuration
- Exploring JavaScript Data Structures
- Designing Your Zone Themes
- Configuring Zone Metadata
- Positioning Zone Labels
- Testing with React DevTools
- Essential Terms
- Ask the AI

☁️ Access Your Codespace

Visit github.com/codespaces to relaunch your Codespace from Session 3.

🖥️ Adding the HUD and Coordinate Display

Let's add the game's **HUD** (Heads Up Display) and a coordinate helper to assist with zone positioning.

1. **Open** `src/App.jsx` and add the new imports at the top:

```
import HUD from "./components/HUD";
import CoordinateDisplay from "./components/CoordinateDisplay";
```

2. **Update the PLAYING screen section** to use a **React Fragment** and include both components:

```
{screen === SCREENS.PLAYING && (
  <>
    <GameMap />
    <HUD />
    <CoordinateDisplay />
  </>
)}
```

3. **Test the changes**: Run `npm run dev` and navigate to the game screen to see the new HUD and coordinate display

💡 Why This Matters

React Fragments (`<> ... </>`) are like invisible containers — they let you snap together multiple components without clutter. React components must return a single root element, so fragments solve this requirement cleanly. The **HUD** shows game progress, while **CoordinateDisplay** helps you position zone labels precisely.

🏆 Bonus Challenge

Try removing the fragment and see what error React gives you when trying to return multiple elements!

📋 Understanding Metadata and Configuration

Now let's understand what makes your game tick — **metadata** and **configuration files** that define your entire game experience.

Metadata is data about data. Think of it like a restaurant menu: it tells you everything about the dish — name, price, ingredients, spice level — but it's not the actual food. Your zone metadata works the same way: it describes each zone's properties without being the actual trivia questions.

💡 Why This Matters

Configuration files like `zones.js` are the backbone of **data-driven architecture** — a professional approach where you separate data from code. Think of your `zones.js` file as the DNA of your game experience: it contains all the genetic information that defines how your

zones look, behave, and connect to trivia content. This makes your app flexible, maintainable, and easy to modify without touching component code.

🏗️ Exploring JavaScript Data Structures

Let's explore the fundamental **data structures** that power your zone configuration — **arrays** and **objects**.

1. **Open** `src/data/zones.js` and examine the `ZONES` array structure
2. **Notice the data types**:
   - **Strings** for text: `"Forest of Knowledge"`
   - **Numbers** for IDs and counts: `18`, `4`
   - **Objects** for complex data: `mapLabel: { x: 225, y: 140 }`
3. **Understand the nesting**: An **array** of **objects**, where each object has **properties** that can be different data types

💡 Why This Matters

Arrays are like playlists — they keep things in order. **Objects** are like contact cards — they store all the details about one thing. Together, they're the perfect combo for organizing your game world and representing complex real-world data in code.

🎨 Designing Your Zone Themes

Now for the creative part — designing three distinctive zone themes that create a cohesive game experience!

1. **Explore available categories**: Visit https://opentdb.com/api_category.php to see all trivia categories
2. **Brainstorm zone concepts** that match the visual environments:
   - **Forest theme**: Nature, animals, science categories
   - **Desert theme**: History, geography, mythology categories
   - **Ice castle theme**: Entertainment, sports, art categories
3. **Plan your zones** — each zone needs:
   - **id**: Zone number (0, 1, 2)
   - **name**: Creative zone title

- **subtitle**: Zone tagline/description
- **categoryId**: Question type from the API categories
- **difficulty**: “easy”, “medium”, or “hard”
- **questionCount**: How many questions (max 50)
- **mapLabel**: Position and styling (we’ll configure this later on)

💡 Why This Matters

Cohesive theming creates an immersive game experience. By matching visual environments with appropriate trivia categories, you create logical connections that help players navigate and remember your game. This attention to **user experience** separates good games from great ones.

⚙️ Configuring Zone Metadata

Time to implement your zone designs by updating the `ZONES` array with your custom metadata.

1. **Open** `src/data/zones.js`
2. **Update the first zone object** with your Zone 0 design:

```
{
  id: 0,
  name: "Your Zone Name",
  subtitle: "Your Zone Subtitle",
  categoryId: 18, // Your chosen category
  difficulty: "easy",
  questionCount: 4,
  mapLabel: {
    x: 225,
    y: 140,
    fontSize: "35",
    fontFamily: "Pirata One, serif",
    color: "#333",
    fontWeight: "normal",
    alignment: "left",
  },
},
```

3. **Add Zone 1 and Zone 2** by copying the structure and updating all properties (keep the mapLabel coordinates for now — we'll position them precisely in the next section)

4. **Save the file** — note that changes to `zones.js` will trigger a full page reload (not HMR-friendly)

💡 Why This Matters

Object properties use colon syntax (`name: "value"`) and are separated by commas. Each zone object contains different **data types**: strings for text, numbers for IDs, and nested objects for complex styling. This structure makes your game data organized and easy to modify.

📍 Positioning Zone Labels

Use the `CoordinateDisplay` component to find optimal positions for your zone labels on the game map.

1. **Navigate to the game screen** and observe the coordinate display
2. **Move your mouse** around the map and note the x, y coordinates — this tool gives you real-time feedback, no guessing, just precision
3. **Find good positions** for each zone label that don't overlap with visual elements
4. **Update the mapLabel coordinates** in your zone objects:

```
mapLabel: {
  x: 225, // Your chosen x coordinate
  y: 140, // Your chosen y coordinate
  // ... other styling properties
}
```

5. **Test each zone** by navigating to the game screen and confirming label placement

💡 Why This Matters

Coordinate positioning requires precise **number** values to place UI elements exactly where you want them. The **CoordinateDisplay** component gives you real-time feedback, making it easy to find perfect positions without guessing.

🏆 Bonus Challenge

Try different `fontSize` and `color` values to customize your zone labels' appearance!

🔍 Testing with React DevTools

Let's use React DevTools to explore your zone configuration and test game progression.

1. **Open DevTools**: Press F12 or right-click → Inspect
2. **Find Components tab**: Look for "Components" next to Console, Network, etc.
3. **Locate GameProvider**: Click on GameProvider in the component tree
4. **Examine the hooks**: Look for the zoneProgress state (if you don't see hook names clearly, click the gear icon and enable "Parse hook names")
5. **Find zoneProgress array**: Under hooks, locate the zoneProgress array
6. **Experiment with progression**: Change the first zone's `completed` property to `true`
7. **Observe state changes**: Notice how `activeZone` and `currentZone` update automatically
8. **Check the HUD**: See how the Context.Provider props reflect the updated game state

💡 Why This Matters

React DevTools lets you manipulate **state** directly to test different game scenarios without playing through the entire game. This is invaluable for debugging and understanding how your **shared state** affects the entire app.

📚 Essential Terms

Quick reference for all the data structure and configuration concepts you just learned:

Term	Definition	Why it matters
📋 metadata	Data that describes other data — information about information.	Your zone configuration describes how to get and display trivia questions without being the questions themselves.
🧮 array	An ordered list of items using bracket syntax `[]` with zero-based indexing.	Perfect for storing your three game zones in a specific order that matches the game progression.

📦 object	A collection of key-value pairs using curly brace syntax `{}` with colon-separated properties.	Ideal for zone properties like name, difficulty, and styling — each zone is an object with multiple attributes.
🔤 string	Text data enclosed in quotes, used for names, descriptions, and categories.	Zone names, subtitles, and difficulty levels are all strings that display to users.
🔢 number	Numeric data without quotes, used for IDs, counts, and coordinates.	Category IDs, question counts, and map coordinates are numbers used for calculations and positioning.
🏷️ property	A key-value pair within an object, accessed using dot notation like `zone.name`.	Each zone object has properties like name, categoryId, and mapLabel that define its characteristics.
⚛️ React Fragment	JSX syntax `<> ... </>` that groups elements without adding extra DOM nodes.	Lets you return multiple components from the PLAYING screen without wrapper divs cluttering your HTML.

🤖 Ask the AI — Data Structure Mastery

You just designed cohesive zone themes, configured complex JavaScript data structures, and experienced data-driven architecture — excellent work!

Now let's deepen your understanding of data structures, configuration patterns, and the relationship between data and UI. Here are the most impactful questions to ask your AI assistant about today's session:

- **What makes arrays and objects different, and when should I use each?**
- **How does nesting data structures help represent complex real-world information?**
- **Why is separating data from code considered a best practice in software development?**
- **What are the benefits of using configuration files like zones.js?**

- **How do React Fragments solve the single root element requirement?**
- **What is metadata and why is it important in application design?**
- **How does the CoordinateDisplay component help with UI positioning?**

💬 Pro Tip:

Data-driven architecture is everywhere in professional development. Think of your zones.js file as the “DNA” of your game — change the data, and the entire game experience changes without touching any component code. This separation makes apps scalable and maintainable.