Session 5 Instructor Guide: Generating Dynamic Questions

Learning Outcomes

By the end of Session 5, students will be able to:

1. **Define APIs** as communication interfaces between applications and explain their role in modern web development
2. **Read API documentation** to understand request parameters and response formats
3. **Explain asynchronous programming** and distinguish it from synchronous code execution
4. **Use async/await and promises** to write readable, maintainable asynchronous JavaScript
5. **Make HTTP requests** using the Fetch API to retrieve data from external sources
6. **Parse JSON responses** and convert them into JavaScript objects and arrays
7. **Build dynamic URLs** using template literals and zone configuration data
8. **Transform API data** into game-ready format using array methods and helper functions
9. **Handle network errors** gracefully with try/catch blocks
10. **Decode URL-encoded text** to convert API responses into readable format
11. **Apply data validation** to ensure API responses are complete and correctly formatted

Instruction

Instructor introduces key concepts students need to succeed:

1. **APIs: The Internet's Communication System** - Define APIs as interfaces that let applications talk to each other, using restaurant menu analogy
2. **OpenTrivia Database Exploration** - Live demo of API endpoint showing raw JSON data and URL encoding
3. **Asynchronous Programming Fundamentals** - Compare synchronous vs asynchronous execution using coffee shop analogy
4. **Fetch API and HTTP Requests** - Introduce fetch as the modern way to request data from servers
5. **JSON: The Universal Data Format** - Explain JSON as the standard format for API communication
6. **Async/Await Syntax** - Show how async/await makes asynchronous code readable and maintainable
7. **Data Transformation Patterns** - Demonstrate the ability to transform API data into the format your application needs
8. **Error Handling and Validation** - Emphasize robust code that handles network failures gracefully
9. **Helper Functions and Managing Complexity** - Show how breaking complex problems into focused functions makes code maintainable and easier to understand

10. **Professional Development Workflow** - Incremental implementation with testing at each step
11. **Let's Connect!** - Launch the hands-on mission: complete the API integration to add dynamic questions to the game

---

Slide Deck Outline

Slide 1: Welcome to the Real Internet! 🌐

- **Title:** "Session 5: Connecting to External APIs — Generating Dynamic Questions"
- **Session 4 Recap:** "Last time: You configured game zones using metadata and JavaScript data structures"
- **Hook:** "Your zones have themes — now let's fill them with real questions from the internet!"
- **Today's Mission:**
  - **Connect** to external APIs for live data
  - **Master** asynchronous programming with async/await
  - **Transform** API responses into game-ready data
  - **Handle** network errors like a professional developer
  - **Experience** the power of real-time data integration
- **Visual:** Data flow diagram showing API → Transform → Game
- **Connection:** "From static configuration to dynamic, internet-powered content!"

Slide 2: APIs - The Internet's Communication System 📡

- **Title:** "What Are APIs and Why Do They Matter?"
- **Restaurant Menu Analogy:**
  - **Menu** = API documentation (tells you what's available)
  - **Order** = HTTP request (asking for specific data)
  - **Food** = JSON response (the actual data you receive)
  - **Waiter** = API endpoint (handles the communication)
- **Real-World Examples:**
  - **Instagram** - Photo API, user API, messaging API
  - **Spotify** - Music streaming API, playlist API, search API
  - **Weather apps** - Forecast API, location API, alerts API
- **Your Game:** OpenTrivia Database API provides thousands of trivia questions
- **Professional Context:** "Modern apps are built by connecting multiple APIs together"
- **Student Preview:** "You'll request real trivia questions and transform them for your game"

Slide 3: OpenTrivia Database - Your Question Source 🎯

- **What is OpenTrivia DB:** Free, open-source trivia question database with thousands of questions across multiple categories
- **API Documentation:** https://opentdb.com/api_config.php - Essential resource for understanding available parameters
- **Key Features:**
  - **Multiple categories** - Science, history, entertainment, sports, and more
  - **Difficulty levels** - Easy, medium, hard
  - **Question types** - Multiple choice, true/false
  - **No API key required** - Free to use for educational projects
- **Live Demo:** Visit API endpoint in browser
- **URL Breakdown:**
  `https://opentdb.com/api.php?amount=3&category=18&type=multiple&difficulty=easy&encode=url3986`
  - **amount=3** - Request 3 questions
  - **category=18** - Computer Science category
  - **difficulty=easy** - Easy difficulty level
  - **encode=url3986** - URL encode special characters
- **Raw JSON Response Analysis:**
  - **response_code: 0** - Success indicator
  - **results array** - Contains the actual questions
  - **URL encoding** - `%20` = space, `%3A` = colon
- **Documentation Skills:** “Reading API docs is essential - they tell you what parameters are available and what responses to expect”
- **The Challenge:** “This API data needs to be transformed into game questions”
- **Student Mission:** “Your job is to fetch this data and transform it”

Slide 4: Synchronous vs Asynchronous - The Coffee Shop Analogy ☕

- **Title:** “Understanding How Code Handles Waiting”
- **Visual:** Split-screen comparison with coffee shop scenarios

Synchronous (Blocking): - **Fast food counter** - One order at a time, everyone waits - **Code behavior** - Each line waits for the previous to complete - **Problem** - App freezes while waiting for network requests

Asynchronous (Non-blocking): - **Coffee shop** - Order, get number, sit down while they prepare - **Code behavior** - Start request, continue other work, handle result when ready - **Benefit** - App stays responsive during network operations

- **Key Insight:** “Network requests take time — asynchronous code keeps your app responsive”

- **Student Connection:** “Your fetch requests will be asynchronous so the game doesn’t freeze”

Slide 5: Fetch API - Modern Data Requests 🚀

- **Title:** “The Modern Way to Request Data”
- **Basic Fetch Syntax:**

```
const response = await fetch(url);
const data = await response.json();
```

- **What Happens:**
  1. **fetch(url)** - Send HTTP request to server
  2. **await** - Wait for response without blocking
  3. **response.json()** - Parse JSON data from response
  4. **data** - JavaScript object ready to use
- **Error Handling:**

```
try {
  const response = await fetch(url);
  const data = await response.json();
} catch (error) {
  console.log("Request failed:", error);
}
```

- **Professional Pattern:** “Always wrap fetch in try/catch for robust error handling”

Slide 6: JSON - The Universal Data Language 📋

- **Title:** “JavaScript Object Notation - How APIs Communicate”
- **What is JSON?**
  - **Text format** that looks like JavaScript objects
  - **Universal standard** for data exchange
  - **Human readable** but structured for machines
- **JSON vs JavaScript Object:**

```
// JSON (text format)
'{"name": "Alice", "age": 25}'

// JavaScript Object (in memory)
{name: "Alice", age: 25}
```

- **Parsing Process:** `JSON.parse()` converts text to objects, `response.json()` does this automatically
- **Why APIs Use JSON:** Language-independent, lightweight, widely supported
- **Student Connection:** "OpenTrivia DB sends JSON — your code converts it to JavaScript objects"

Slide 7: Async/Await - Making Asynchronous Code Readable 🎭

- **Title:** "Async/Await: Asynchronous Code That Looks Synchronous"
- **The Problem with Callbacks:**

```
// Hard to read and debug
fetch(url).then(response => {
  return response.json();
}).then(data => {
  console.log(data);
}).catch(error => {
  console.log(error);
});
```

- **The Async/Await Solution:**

```
// Easy to read and debug
async function fetchData() {
  try {
    const response = await fetch(url);
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.log(error);
  }
}
```

- **Key Rules:**
  - **async** keyword before function declaration
  - **await** keyword before asynchronous operations
  - **try/catch** for error handling
- **Student Benefit:** "Your API code will be clean and easy to understand"

Slide 8: Data Transformation - API to Game Format 🔄

- **Title:** "Transforming API Data for Your Game"
- **Visual:** Before/After comparison showing data transformation

API Response (Raw Format):

```
{
  "question": "What%20does%20GHz%20stand%20for%3F",
  "correct_answer": "Gigahertz",
  "incorrect_answers": ["Gigahotz", "Gigahetz", "Gigahatz"]
}
```

Game Format (Transformed):

```
{
  "question": "What does GHz stand for?",
  "answers": ["Gigahotz", "Gigahertz", "Gigahetz", "Gigahatz"],
  "correct": 1
}
```

- **Transformation Steps:**
  1. **Decode** URL-encoded text (`%20` → space)
  2. **Combine** correct and incorrect answers
  3. **Shuffle** answer order randomly
  4. **Find** correct answer index in shuffled array
- **Professional Pattern:** “APIs rarely return data in exactly the format you need”

Slide 9: Helper Functions - Managing Complexity 🧩

- **Title:** “Breaking Complex Problems into Manageable Pieces”
- **Core Concept:** “Decompose complex tasks into smaller, focused functions that can be combined”
- **Helper Function Benefits:**
  - **Reusability** - Write once, use multiple times
  - **Maintainability** - Changes in one place update everywhere
  - **Testability** - Easy to test small, focused functions
  - **Readability** - Clear function names explain what code does
  - **Complexity Management** - Tackle big problems by solving smaller ones
- **Today’s Helpers:**
  - **buildApiUrl()** - Constructs request URL from zone data
  - **decodeText()** - Converts URL encoding to readable text
  - **shuffleAnswers()** - Randomizes answer order
  - **transformQuestion()** - Converts API format to game format
- **Professional Insight:** “Good developers break complex problems into simple, composable pieces”

Slide 10: Error Handling - Building Robust Applications 🛡️

- **Title:** “Planning for When Things Go Wrong”
- **Common API Failures:**
  - **Network errors** - Internet connection issues
  - **Server errors** - API temporarily down
  - **Invalid responses** - Empty or malformed data
  - **Rate limiting** - Too many requests too quickly
- **Error Handling Strategy:**

```
try {
  const response = await fetch(url);
  const data = await response.json();

  if (!data.results || data.results.length === 0) {
    return []; // Handle empty response
  }

  return processData(data);
} catch (error) {
  console.log("Failed to fetch questions:", error);
  return []; // Return safe fallback
}
```

- **Professional Mindset:** “Always assume external services might fail”
- **Student Application:** “Your game will gracefully handle network issues”

Slide 11: Fetch Dynamic Trivia Questions! 🚀

- **Today’s Mission: Connect your game to the internet**
  1. **Replace alert** with basic fetch request and response logging
  2. **Add data validation** to handle empty API responses
  3. **Implement transformQuestion** step-by-step with console logging
  4. **Complete fetchQuestions** integration with full data transformation
  5. **Test API integration** using React DevTools state inspection
  6. **Clean up debugging code** for production-ready implementation
- **Success Criteria:**
  - Zones load real trivia questions from OpenTrivia DB
  - Questions are properly decoded and formatted
  - Answers are shuffled with correct index tracking
  - Error handling prevents crashes on network failures

- **Professional Workflow:** “Build incrementally, test frequently, handle errors gracefully”

[HANDS-ON WORK HAPPENS HERE]

Slide 12: Data Flow Architecture - The Complete Picture 🔄

- **Title:** “Tracing Data from Click to Questions”
- **Visual:** Complete data flow diagram

```
User clicks zone → GameMap handleZoneClick → GameContext loadQuestionsForZone
                                                        ↓
React state updates ← Clean game objects ← transformQuestion ← API response
                                                        ↓
                                                trivia.js fetchQuestions
                                                        ↓
                                                OpenTrivia Database API
```

- **Key Components:**
  - **GameMap** - Handles user interaction
  - **GameContext** - Manages application state
  - **trivia.js** - Handles API communication and data transformation
  - **OpenTrivia DB** - External data source
- **Professional Architecture:** “Separation of concerns — each component has a specific responsibility”
- **Student Achievement:** “You built a complete data pipeline from user interaction to external API”

Slide 13: What’s Next - Caching and Performance 💾

- **Title:** “Preview of Session 6”
- **Today’s Achievement:** “You connected your game to real internet data with complete API integration”
- **Next Challenge:** “Add caching to make your game faster and more efficient”
- **Concepts Coming:**
  - **Browser storage** - Save API responses locally
  - **Cache strategies** - When to use cached vs fresh data
  - **Performance optimization** - Reduce unnecessary network requests
  - **Cache management** - Clear old data when needed
- **Motivation:** “Your questions will load instantly after the first request!”
- **Visual:** Performance comparison showing cached vs uncached request times