D1 Enhanced Framework Sample

This document demonstrates the D1 Enhanced Framework with strategic callouts and verification patterns.

🔄 Transforming API Data

The API returns data in its own format, but your game needs a different structure. You'll build the transformation incrementally, testing after each step to see your progress.

Step 1: Set Up Transformation Testing

Before building the transformation, add logging to see the before and after.

- **Add** after the validation check in fetchQuestions:

```
const firstQuestion = data.results[0];
console.log("Before transform:", firstQuestion);

const transformed = transformQuestion(firstQuestion);
console.log("After transform:", transformed);
```

- **Verify:** Click a zone and check the console. You'll see `undefined` because transformQuestion isn't implemented yet.

Step 2: Extract the Raw Properties

Pull out the question and answers from the API response.

- **Add** in the transformQuestion function:

```
function transformQuestion(apiQuestion) {
  const question = apiQuestion.question;
  const incorrectAnswers = apiQuestion.incorrect_answers;
  const correctAnswer = apiQuestion.correct_answer;

  console.log("Extracted properties:", { question, incorrectAnswers, correctAnsw
}
```

- **Verify:** Click a zone. You should see the raw extracted data in the console.

Step 3: Decode the URL Encoding

Convert the encoded text (`What%20does%20GHz%20stand%20for%3F`) into readable format.

- **Update** transformQuestion to use the helper function:

```
function transformQuestion(apiQuestion) {
  const question = decodeText(apiQuestion.question);
  const incorrectAnswers = apiQuestion.incorrect_answers.map(answer => decodeTex
  const correctAnswer = decodeText(apiQuestion.correct_answer);

  console.log("Decoded data:", { question, incorrectAnswers, correctAnswer });
}
```

- **Expected output:**

```
{
  "question": "What does GHz stand for?",
  "incorrectAnswers": ["Gigahotz", "Gigahetz", "Gigahatz"],
  "correctAnswer": "Gigahertz"
}
```

> 💡 **Concept:** The `map()` method transforms each item in an array. Here, it decodes each incorrect answer.

Step 4: Shuffle Answers and Find the Correct Index

Randomize answer order so players can't memorize positions.

- **Add** the shuffling logic:

```
function transformQuestion(apiQuestion) {
  const question = decodeText(apiQuestion.question);
  const incorrectAnswers = apiQuestion.incorrect_answers.map(answer => decodeTex
  const correctAnswer = decodeText(apiQuestion.correct_answer);
  const shuffledAnswers = shuffleAnswers(correctAnswer, incorrectAnswers);
  const correctIndex = shuffledAnswers.indexOf(correctAnswer);

  console.log("Shuffled answers:", shuffledAnswers);
  console.log("Correct answer is at index:", correctIndex);
}
```

- **Verify:** You should see shuffled answers and the index where the correct answer ended up.

> ⚠️ **Warning:** Don't modify the original arrays. The helper functions return new arrays to avoid side effects.

Step 5: Return the Game Object

Build the final format your game needs.

- **Add** the return statement:

```
function transformQuestion(apiQuestion) {
  const question = decodeText(apiQuestion.question);
  const incorrectAnswers = apiQuestion.incorrect_answers.map(answer => decodeTex
  const correctAnswer = decodeText(apiQuestion.correct_answer);
  const shuffledAnswers = shuffleAnswers(correctAnswer, incorrectAnswers);
  const correctIndex = shuffledAnswers.indexOf(correctAnswer);

  return {
    question: question,
    answers: shuffledAnswers,
    correct: correctIndex
  };
}
```

- **Expected output:**

```
{
  "question": "What does CPU stand for?",
  "answers": [
    "Central Process Unit",
    "Computer Personal Unit",
    "Central Processing Unit",
    "Central Processor Unit"
  ],
  "correct": 2
}
```

Step 6: Apply to All Questions

Now use your transformation function to process the entire array.

- **Replace** the test logging in fetchQuestions:

```
const questions = data.results.map(apiQuestion => transformQuestion(apiQuestion)
console.log("All transformed questions:", questions);
return questions;
```

- **Verify the complete implementation:**
  1. Click a zone
  2. Open React DevTools (F12)
  3. Navigate to Components tab
  4. Find GameProvider
  5. Check currentQuestions state
  6. Confirm it contains an array of properly formatted questions

> ✅ **Success Check:** - [ ] Questions array appears in GameProvider state - [ ] Each question has `question`, `answers`, and `correct` properties - [ ] Answers are shuffled (different order each time) - [ ] No console errors

Data transformation is core to web development. APIs rarely return data in your exact format, so you build functions that bridge the gap.

🏆 Adding Score Tracking

Your game needs to track player performance and display it prominently.

Step 1: Add Score State

Create state to track the player's current score.

- **Open** `src/context/GameContext.jsx` and add inside GameProvider:

```
const [score, setScore] = useState(0);
```

- **Add** to the Context value:

```
<GameContext.Provider value={{
  // GAME STATE
  screen,
  score,
  zoneProgress,
  // ... rest of existing properties
```

- **Verify:** Open React DevTools → GameProvider → hooks → score should be 0

> ℹ️ **Note:** State initialized to 0 ensures the game starts with a clean slate.

Step 2: Display Score in HUD

Show the score to players during gameplay.

- **Open** `src/components/HUD.jsx` and add at the top:

```
function Scoreboard() {
  const { score } = useGame();
  return <div className="score-display">Score: {score}</div>;
}
```

- **Update** HUD component to render both components:

```
return (
  <>
    <Scoreboard />
    <CurrentZone />
  </>
);
```

- **Verify:** Navigate to game screen. “Score: 0” should appear in the HUD.

> 🎯 **Pro Tip:** Use React DevTools to change the score value and watch the UI update in real-time.

Step 3: Update Score on Correct Answers

Reward players with points for correct answers.

- **Find** the `recordCorrectAnswer` function in GameContext.jsx
- **Add** point reward:

```
const recordCorrectAnswer = () => {
  setCorrectAnswers((prev) => prev + 1);
  setScore((prev) => prev + POINTS_PER_CORRECT);
};
```

- **Verify:** Answer questions correctly. Score should increase by 100 points each time.

> 💡 **Concept:** Updater functions like `setScore((prev) => prev + 100)` ensure accurate calculations even when React batches multiple state updates.

Callout Type Examples

This section demonstrates all callout types for reference.

Concept Callout

> 💡 **Concept:** Use this for explaining how or why something works.

Warning Callout

> ⚠️ **Warning:** Use this to prevent common mistakes or highlight important gotchas.

Success Check Callout

> ✅ **Success Check:** Use this for verification checklists with multiple conditions.

Error Callout

> ❌ **Error:** Use this for error messages or things to avoid.

Note Callout

> ℹ️ **Note:** Use this for additional context, tips, or side information.

Pro Tip Callout

> 🎯 **Pro Tip:** Use this for advanced techniques or shortcuts.

Verification Pattern Examples

Pattern A: Inline Verification (Simple)

- **Update** the title:

  ```
  <title>Wizcamp Realms - Legends of Trivia</title>
  ```

  Verify: Browser tab should display the new title

Pattern B: Bullet Verification (Standard)

- **Update** the title:

  ```
  <title>Wizcamp Realms - Legends of Trivia</title>
  ```

- **Verify:** Check the browser tab. It should display “Wizcamp Realms - Legends of Trivia”

Pattern C: Dedicated Verification Section (Complex)

- **Update** the scoring system
- **Verify the implementation:**
  1. Start the game
  2. Answer a question correctly
  3. Check that score increases by 10

4. Answer incorrectly
5. Verify score doesn't go below 0

Pattern D: Expected Output Block (API/Data)

- **Fetch** questions from the API:

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

- **Expected output:**

```
{
  "response_code": 0,
  "results": [
    {
      "question": "What does CPU stand for?",
      "correct_answer": "Central Processing Unit"
    }
  ]
}
```

Pattern E: Success Criteria Callout (Checklist)

- **Complete** the scoring system implementation

> ✅ **Success Check:** - [ ] Score displays "0" when game starts - [ ] Score increases by 10 for correct answers - [ ] Score decreases by 10 for incorrect answers (minimum 0) - [ ] Score resets to 0 when "Play Again" is clicked