Session 6 — Adding Question Caching

Browser Storage & Caching 💾

You're about to supercharge your trivia game with browser storage and caching! This guide walks you through implementing localStorage to store API responses, understanding performance optimization techniques, and building a robust caching system that makes your game lightning-fast. Ready to experience the magic of instant loading? Let's go!

Table of Contents

☁️ Accessing Your Codespace

Visit github.com/codespaces to relaunch your Codespace from Session 5.

⚡ Why Caching Matters

Before we dive into code, let's understand why **caching** is crucial for modern web applications.

The Problem: Every time you click a zone, your game makes a network request to the OpenTrivia Database. This means:

- **Slow loading** - Network requests take time
- **Wasted bandwidth** - Downloading the same questions repeatedly

- **Rate limiting** - APIs limit how often you can request data (OpenTrivia allows one request per 5 seconds)
- **Poor user experience** - Users wait for content they've already seen

The Solution: Store API responses locally in the browser so subsequent requests are instant.

Caching is a fundamental performance optimization technique used in some form by every major website and app. When you visit YouTube, Netflix, or Instagram, they cache images, videos, and data locally so your experience is fast and smooth.

🗄️ Exploring Browser Storage

Let's understand **localStorage** — your browser's built-in storage system for saving data locally.

localStorage is like a digital filing cabinet in your browser where you can store **key-value pairs** of information. Unlike temporary data that vanishes when pages refresh, localStorage data survives browser restarts and persists until explicitly removed.

Common localStorage Use Cases:

- **User preferences** - Theme, language, font size
- **Game progress** - Completed levels, high scores, settings
- **Form data** - Draft messages, shopping cart contents
- **API responses** - Cached data for faster loading

localStorage Lifecycle (CRUD Operations):

```
┌─────────────────────────────────────────────────────────┐
│                  localStorage Lifecycle                 │
├─────────────────────────────────────────────────────────┤
│ CREATE/UPDATE: localStorage.setItem('key', 'value')     │
│ READ:          const value = localStorage.getItem('key')│
│ DELETE:        localStorage.removeItem('key')           │
│ CHECK:         if (localStorage.getItem('key')) { ... } │
└─────────────────────────────────────────────────────────┘
```

localStorage provides persistent storage that survives browser refreshes and even computer restarts. It's **synchronous** (blocking), and ideally with small to medium amounts of data. For your trivia game, it's perfect for caching question sets that are relatively small but expensive to fetch over the network.

🔧 Building Cache Helper Functions

Time to build the foundation of your caching system! Helper functions abstract the complexity of localStorage operations into clean, reusable pieces. Each function has a specific role:

Function	Purpose	Key Operation
`getCacheKey`	Creates unique identifiers	Generates consistent cache keys
`getCachedQuestions`	Retrieves stored data	**Deserialization** with `JSON.parse()`
`setCachedQuestions`	Stores new data	**Serialization** with `JSON.stringify()`

1. **Open** `src/services/trivia.js`
2. **Add all three cache helper functions** after the existing helper functions

```
function getCacheKey(zoneId) {
  return `trivia_questions_zone_${zoneId}`;
}

function getCachedQuestions(zoneId) {
  const cacheKey = getCacheKey(zoneId);
  const cached = localStorage.getItem(cacheKey);
  return cached ? JSON.parse(cached) : null;
}

function setCachedQuestions(zoneId, questions) {
  const cacheKey = getCacheKey(zoneId);
  localStorage.setItem(cacheKey, JSON.stringify(questions));
}
```

Example Usage:

```
// This is what your functions do:
setCachedQuestions(0, questions);  // Store questions for zone 0
const cached = getCachedQuestions(0);  // Get questions for zone 0 (or null if none)
```

Notice the **ternary operator** `cached ? JSON.parse(cached) : null` in `getCachedQuestions` — this concise syntax means "If cached data exists, parse it; otherwise return null."

These helper functions represent a fundamental software engineering principle: **abstraction**. By wrapping localStorage complexity in simple functions, you're building the same kind of modular, maintainable code architecture used in modern applications. This pattern makes your caching system easy to test, debug, and extend.

🔄 Updating fetchQuestions with Caching

Now let's integrate your cache functions into the main `fetchQuestions` function to implement the complete caching flow.

1. **Add cache checking** at the beginning of `fetchQuestions` (before the zone lookup):

```
export async function fetchQuestions(zoneId, count = null) {
  // Cache check pattern: try cache first, fetch on miss
  const cachedQuestions = getCachedQuestions(zoneId);
  if (cachedQuestions) {
    console.log(`Cache hit for zone ${zoneId}`);
    return cachedQuestions;
  }

  console.log(`Cache miss for zone ${zoneId} - fetching from API`);

  const zone = getZoneById(zoneId);
  // ... rest of existing code
```

2. **Add cache storage** after successful data transformation (before the return statement):

```
const questions = data.results.map(apiQuestion => transformQuestion(apiQuestion)

setCachedQuestions(zoneId, questions);

return questions;
```

This implements the classic **cache-aside pattern** used in modern applications: check cache first, fetch from source on miss, store result in cache. The console logging helps you understand when cache hits and misses occur, which is valuable for debugging and performance monitoring.

🧪 Testing Your Cache

Let's see your caching system in action! You'll observe cache **misses**, **hits**, and **persistence** using DevTools and localStorage. Understanding this flow is crucial:

```
User clicks zone → Check cache → Cache hit? → Return cached data
                                      |
                                      ▼ (Cache miss)
                                Fetch from API → Store in cache → Return data
```

🔍 Setup: Open DevTools and Locate Local Storage

- **Press** F12 or right-click → Inspect
- **Navigate** to
  - **Chrome/Edge**: Application tab
  - **Firefox**: Storage tab
- In the sidebar, **expand** Local Storage and select your site's domain (e.g., `http://localhost:5173`)
- **Keep** DevTools open as you'll watch cache entries appear in real-time

---

🆕 First-Time Load: Observe a Cache Miss

- **Click** the active zone for the first time
- In the console, **look** for `Cache miss for zone X - fetching from API`
- In localStorage, **confirm**
  - A new entry appears: `trivia_questions_zone_0`
  - It contains serialized JSON data
- **Click** the entry to inspect the cached questions

🔁 Repeat Load: Confirm a Cache Hit

- **Click** the same zone again
- In the console, **look** for `Cache hit for zone X`
- In localStorage, **verify**
  - The entry remains unchanged
  - No new data was fetched

🔄 Page Reload: Test Cache Persistence

- **Refresh** the browser
- **Click** the same zone again
- **Confirm**
  - Console still shows `Cache hit`

- Cached entry is still present in localStorage

🧹 Manual Clear: Test Cache Reset

- In localStorage, **right-click** the cache entry → Delete
- **Click** the zone again
- **Confirm**
  - Console shows `Cache miss`
  - Entry repopulates with fresh data

You're basically becoming a detective! By watching console logs, peeking into browser storage, and tracking network requests, you're learning to **follow the digital breadcrumbs** your code leaves behind. This is exactly how real developers figure out why apps crash, why websites load slowly, or why that "it worked yesterday" bug suddenly appeared. These debugging superpowers will make you unstoppable when building your own projects.

📚 Essential Terms

Quick reference for all the caching and browser storage concepts you just learned:

Term	Definition	Why it matters
⚡ caching	Storing frequently accessed data in fast storage to avoid expensive operations like network requests.	Makes your game feel instant and responsive by eliminating repeated API calls for the same questions.
🚦 rate limiting	API restrictions on request frequency to prevent server overload and ensure fair usage.	OpenTrivia Database limits requests to once every 5 seconds — caching helps avoid these limits.

💾 localStorage	Browser storage that persists data as key-value pairs across sessions and page refreshes.	Your trivia questions stay cached even after closing and reopening the browser, providing instant loading.
🔑 key-value pairs	Data storage format where each piece of information has a unique identifier (key) and associated data (value).	localStorage uses this format: your cache keys identify zones, values contain question data.
📦 serialization	Converting JavaScript objects into text format for storage using JSON.stringify().	localStorage only stores strings, so your question objects must be serialized before storage.
🔄 deserialization	Converting stored text back into JavaScript objects using JSON.parse().	Transforms cached text back into usable question objects for your game.
🎯 cache hit	When requested data is found in cache and can be returned immediately without external requests.	Your zones load instantly on subsequent clicks, providing smooth user experience.
🔍 cache miss	When requested data is not in cache and must be fetched from the original source.	Triggers API request to OpenTrivia Database and stores result for future cache hits.

❓ ternary operator	Concise conditional syntax using ? and : for simple if/else logic in expressions.	Used in your cache retrieval: `cached ? JSON.parse(cached) : null` — clean and readable.

🤖 Ask the AI — Adding Question Caching

You just built your first caching system with localStorage and helper functions — great work!

Here are some key questions to ask your AI assistant to deepen your understanding of what you just built:

- **Why is caching so important for web apps and user experience?**
- **What happens when I use JSON.stringify and JSON.parse with my question data?**
- **Why do we need helper functions like getCacheKey and getCachedQuestions?**
- **What would happen if I didn't have caching in my trivia game?**
- **How can I use the browser DevTools to debug localStorage issues?**