ADR-CS-6310-A3-T38-001: Enhanced Pokémon Platform Architecture

Team #38 Members: Charles Michael Sgro, Hugh Gorman, Jeong "Julie" Weon, Seung, Hong, Surendranath "Suren" Gujjala.

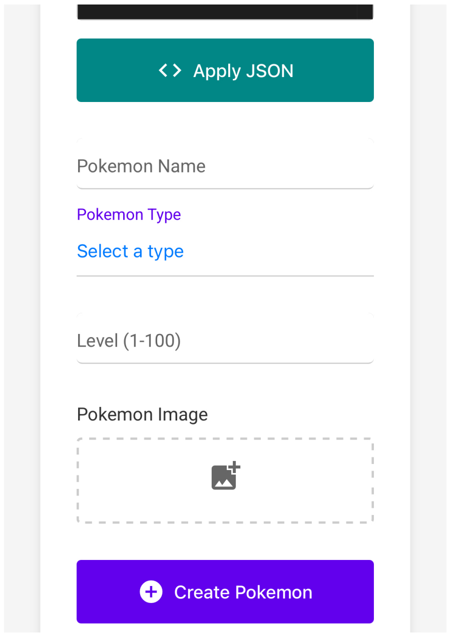
Proposed Functional Enhancements

Create Persistent Pokémon

1. The front-end web UI will provide validated, reactive forms for uploading JSON payloads to create unique Pokémon entries.
    - How do we define uniqueness?
        - Pokémon name
2. Stateful rendering of data using database and browser-based caching (IndexedDB/LocalStorage).
3. REST API endpoints with validated and documented request/response models
4. Battle state management using persistent storage (PostgreSQL)
5. **Draft of a JSON Schema of a Pokémon:**

```
export const pokemonJsonSchema = {
 type: "object",
 properties: {
  name: { type: "string", minLength: 3 },
  type: { type: "string", enum: ["fire", "water", "grass", "electric", "psychic"] },
  level: { type: "integer", minimum: 1, maximum: 100 },
  image: { type: "string" },
  hp: { type: "integer", minimum: 1, maximum: 255 },
  hpSkills: {
   type: "array",
   items: {
    type: "string",
    enum: ["Sturdy", "Regenerator", "Overgrow", "Blaze", "Torrent", "Solar Power", "Speed Boost", "Wonder Guard"]
   }
  }
 },
 required: ["name", "type", "level"],
 additionalProperties: false
};
```

- The form will be based on this JSON schema (not all fields shown in mock-up). Each data input will have an appropriate form control. For example, dropdowns for lookup values and text boxes for names.
- A pre-set list will be stored in IndexedDB via Dexie and mirrored in a PostgreSQL lookup table.
- Each Pokémon will have standardized HP values.
- Example form (non-inclusive of all of the aforementioned)



Proposed Non-Functional Enhancements

Archivability

- Battle/Pokémon history statistics stored in a metrics table
- Battle History Logs table (PostgreSQL) with archive flag column, defined retention and truncation periods
- Battle history automatically archived after 5 battles
- Battle History Logs reporting interface with 1-week retention period

Performance

1. **Caching Layer**

    - We will use Dexie, an IndexedDB wrapper library, to store application state in the browser with SQL-like syntax
    - Expected performance improvement: minimum 2 times faster compared to direct PostgreSQL queries
    - We will implement a caching layer using IndexedDB. The client will store key responses and refresh only when the cache expires. Our final report will include benchmark comparisons demonstrating performance improvements in load times and API usage.
        - What are we caching?
            - Pokémon, Pokémon Skills (referential data), Battle History
        - Metrics will include render time, number of API calls and cache hit/miss ratio.
        - Update Cache values as cached entities are mutated
            - Store a compare and swap value in Unix epoch time to determine cache validity
            - After the 1-day caching TTL expires, Battle/Pokémon History Statistics will be archived
        - Performance Metric Measurement Mechanism
            - How: Postman
                - Create a Collection with our API schema imported
                - Benchmark each cached API route with caching enabled and disabled; generate performance KPI reports

Technical Implementation

We will use NextJS with TypeScript, PostgreSQL with Prisma ORM, and Docker for containerization. This stack provides:

- Frontend Benefits
    - Modular component architecture
    - Responsive user interface
- Backend Benefits
    - Type-safe API development

- Efficient database operations
- Robust error handling

Status: **Pending** review by **Eric C. Weinberg, TA**

Date: rev. 4 2025-04-05