This plan is designed to be actionable and can be provided to a development team or an AI agent for implementation. It removes the need for user authentication and a backend database for this initial phase, focusing entirely on creating a functional and visually appealing client-side experience using a local dataset for the Pokémon Base Set.

Pokémon TCG Collector: Simplified MVP Frontend Build Plan

Version: 1.1
Objective: To construct a client-side-only MVP of the Pokémon TCG Collector web app. This version will be a static application that showcases the core browsing functionality using only the original Pokémon TCG Base Set, with local images and data.

1. Simplified MVP Goals

- **No User Login:** The app will operate without user accounts. Collection tracking will be handled via the browser's localStorage.
- **Frontend First:** This build plan is exclusively for the frontend. No backend or database setup is required.
- **Focused Dataset:** The app will only display data and images for the original **Pokémon Base Set**. All other sets will be visually represented but marked as "Coming Soon."
- **Local Assets:** All card images will be sourced from a local \images\pokemon_base_set directory within the project structure. Cards from other sets will use a standardized placeholder image.

2. Technology Stack (Frontend)

- **Framework: React** (using Create React App for rapid setup). Its component-based architecture is ideal for this project.
- **Styling: Styled-Components** or **Tailwind CSS**. This allows for easy implementation of the Style Guide on a per-component basis.
- **State Management:** React Hooks (useState, useEffect, useContext) will be sufficient for managing the state of this simplified MVP.

3. Project Structure (File Layout)

A clear file structure is crucial for scalability.

```
/src

|-- /assets
| |-- /fonts
| |-- /icons
| |-- /images
| |-- /pokemon_base_set
| |-- 1.png
| |-- 2.png
| |--... (all 102 card images)
| |-- placeholder.png
|-- /components
| |-- /Card
| | |-- Card.js
| | |-- Card.styles.js
| |-- /CardGrid
| | |-- CardGrid.js
| | |-- CardGrid.styles.js
| |-- /Header
| | |-- Header.js
| | |-- Header.styles.js
| |-- /SetBrowser
| | |-- SetBrowser.js
| | |-- SetBrowser.styles.js
|-- /data
| |-- sets.js        // Metadata for all TCG sets
| |-- base_set_cards.js // Detailed data for Base Set cards
|-- /hooks
| |-- useLocalStorage.js // Custom hook for collection management
|-- App.js
|-- index.js
|-- index.css          // Global styles, font imports
```

4. Data Structure (Local JSON/JS Files)

All data will be hardcoded in JavaScript files for the MVP.

/data/sets.js

This file will list all TCG series and sets to populate the browser UI.

JavaScript

```javascript
export const tcgSeries =
 },
 //... more series
];
```

/data/base_set_cards.js

This file contains the detailed data for each card in the Base Set.

JavaScript

```javascript
export const baseSetCards =
   price_sgd: 337.50
 },
 //... all 102 cards
```

```
];
```

5. Component Hierarchy & Wireframes (Mermaid Diagram)

This diagram shows how the UI components will be structured.

Code snippet

```
graph TD
    A[App] --> B[Header];
    A --> C;
    A --> D[CardGrid];
    D --> E[Card];

    subgraph "User Interaction"
        E -- Click --> F;
    end

    style A fill:#f9f,stroke:#333,stroke-width:2px
    style F fill:#bbf,stroke:#333,stroke-width:2px
```

Wireframes (Text-Based)

1. Main App View (App.js)

```
+-------------------------------------------------------------------------+

| Header Component |
| |
+-------------------------------------------------------------------------+
| Set Browser Component |
|... |
| |
|... |
+-------------------------------------------------------------------------+

| Card Grid Component (showing cards from selected set) |
| |
| +-----------+  +-----------+  +-----------+  +-----------+  +-----------+ |
| | Card Comp | | Card Comp | | Card Comp | | Card Comp | | Card Comp | |
| | [Image] | | [Image] | | [Image] | | [Image] | | [Image] | |
| | Name/# | | Name/# | | Name/# | | Name/# | | Name/# | |
| | [+ Add] | | [✓ Added] | | [+ Add] | | [+ Add] | | [✓ Added] | |
| +-----------+  +-----------+  +-----------+  +-----------+  +-----------+ |
| |... more cards... |
+-------------------------------------------------------------------------+
```

2. Card Component (Card.js)

```
// State: Added (✓)
+-----------------------+

| |
| [ Card Image ] |
| (from /images/...) |
| |
+-----------------------+

| Charizard 4/102 |
| Rare Holo |
| [✓ Added to Collection] |
+-----------------------+
```

3. Card Detail Modal (on card click)

```
+------------------------------------------------------------------------+

| [X] Close |
| +--------------------------+   Charizard - 4/102 ★ |
| | | Set: Base Set |
| | | Type: Fire |
| | [ Large Card Image ] | HP: 120 |
| | | ---------------------------------------- |
| | | Market Value (Ungraded): |
| | | $250.00 USD / ~$337.50 SGD |
| +--------------------------+   ---------------------------------------- |
| |
| Your Collection: |
| +------------------------------------------------------------------+ |
| | Variant | Condition | Qty | | |
| |-----------|-----------|-----|------------------------------------| |
| | Unlimited | Near Mint | [1] | [Add] | |
| +------------------------------------------------------------------+ |
| |
+------------------------------------------------------------------------+
```

6. Core Logic (Pseudo Code in React)

This pseudo code outlines the primary frontend logic.

/App.js - Main Component

JavaScript

```
import React, { useState, useEffect } from 'react';
import Header from './components/Header';
import SetBrowser from './components/SetBrowser';
import CardGrid from './components/CardGrid';
import { baseSetCards } from './data/base_set_cards';
import useLocalStorage from './hooks/useLocalStorage';

function App() {
 const [allCards, setAllCards] = useState();
 const [filteredCards, setFilteredCards] = useState();
 const [collection, setCollection] = useLocalStorage('userCollection', {});

 // Load card data on initial render
 useEffect(() => {
  setAllCards(baseSetCards);
  setFilteredCards(baseSetCards);
 },);

 // Function to handle search
 const handleSearch = (searchTerm) => {
  if (!searchTerm) {
   setFilteredCards(allCards);
  } else {
   const lowercasedTerm = searchTerm.toLowerCase();
   const results = allCards.filter(card =>
    card.name.toLowerCase().includes(lowercasedTerm)
   );
   setFilteredCards(results);
  }
 };

 // Function to add/remove card from collection (simplified)
 const toggleCardInCollection = (cardId) => {
  const newCollection = {...collection };
  if (newCollection[cardId]) {
   delete newCollection[cardId]; // Remove if exists
  } else {
   newCollection[cardId] = { quantity: 1, condition: 'NM' }; // Add if not
  }
  setCollection(newCollection);
 };

 return (
  <div>
   <Header onSearch={handleSearch} />
   <SetBrowser />
   <CardGrid
```

```
      cards={filteredCards}
      collection={collection}
      onToggleCard={toggleCardInCollection}
    />
   </div>
 );
}
```

/components/Card.js - Individual Card Display

JavaScript

```
import React from 'react';

function Card({ cardData, isCollected, onToggle }) {
 const { name, number, rarity, imageUrl } = cardData;

 return (
   <div className="card-container">
    <img src={imageUrl} alt={name} />
    <div className="card-info">
      <p>{name} {number}</p>
      <p>{rarity}</p>
    </div>
    <button onClick={() => onToggle(cardData.id)}>
      {isCollected? '✓ Added' : '+ Add to Collection'}
    </button>
   </div>
 );
}
```

/hooks/useLocalStorage.js - Custom Hook for Persistence

JavaScript

```javascript
import { useState, useEffect } from 'react';

function useLocalStorage(key, initialValue) {
  // Get from local storage then
  // parse stored json or return initialValue
  const = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item? JSON.parse(item) : initialValue;
    } catch (error) {
      console.log(error);
      return initialValue;
    }
  });

  // Return a wrapped version of useState's setter function that...
  //... persists the new value to localStorage.
  const setValue = (value) => {
    try {
      const valueToStore = value instanceof Function? value(storedValue) : value;
      setStoredValue(valueToStore);
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.log(error);
    }
  };

  return [storedValue, setValue];
}
```