▯ Frontend Interview Task – Senior React Engineer

📊 Overview

Build a **responsive Pokémon browser** using **React + TypeScript**. The application must include:

- A **grid view** of Pokémon with two variations:
    - One with **pagination controls**
    - One with a **"Load More"** button
- A **dedicated detail page** for each Pokémon
- Proper **loading, error**, and **responsive UI** behavior

This task is timeboxed to **4 hours** and focuses on clean structure, UI precision, and real-world usability.

📄 Tech Stack & Tools

- React
- **TypeScript** (mandatory)
- Any styling approach (e.g., Tailwind CSS, CSS Modules, Styled Components)
- Deployment via **Vercel**, **Netlify**, or **Cloudflare Pages**
- Git with clean, meaningful commit history

🔗 API Reference

Use the public PokéAPI to fetch and display Pokémon data.

Required Endpoints:

- **List Pokémon** (paginated):

GET <https://pokeapi.co/api/v2/pokemon?limit=10&offset=0>

- **Get single Pokémon details**:

GET <https://pokeapi.co/api/v2/pokemon/{id}>

🔢 Requirements

1. Pokémon List Views

Implement two separate views for displaying Pokémon:

- **Pagination View**

    - Show a grid of Pokémon cards (name + sprite)
    - Include pagination controls (page numbers + next/previous)
- **Load More View**
    - Use a "Load More" button to append the next batch of Pokémon
    - Avoid duplicates or state conflicts

2. Detail Page

Clicking a Pokémon must navigate to a **dedicated detail page** that displays:

- Name
- Sprite
- Height
- Weight
- Types

This must be a **separate route** and not a modal, drawer, or inline expansion.

3. State Handling

- While fetching data:
    - If the **design includes a loading state**, implement it as shown
    - If not, use a **skeleton** or **spinner/indicator** as appropriate
- On failure:
    - Display an **error message**
    - Include a **retry option**

4. Responsiveness

- The app must be **fully responsive**
- It should render and function correctly on:
    - **Desktop**
    - **Tablet**
    - **Mobile**
- Grid layouts should adapt gracefully across breakpoints

5. Code Quality

- Organize your code into **modular, testable** components

- Use **separation of concerns** (API calls, views, components)
- No tests required, but structure must support testability

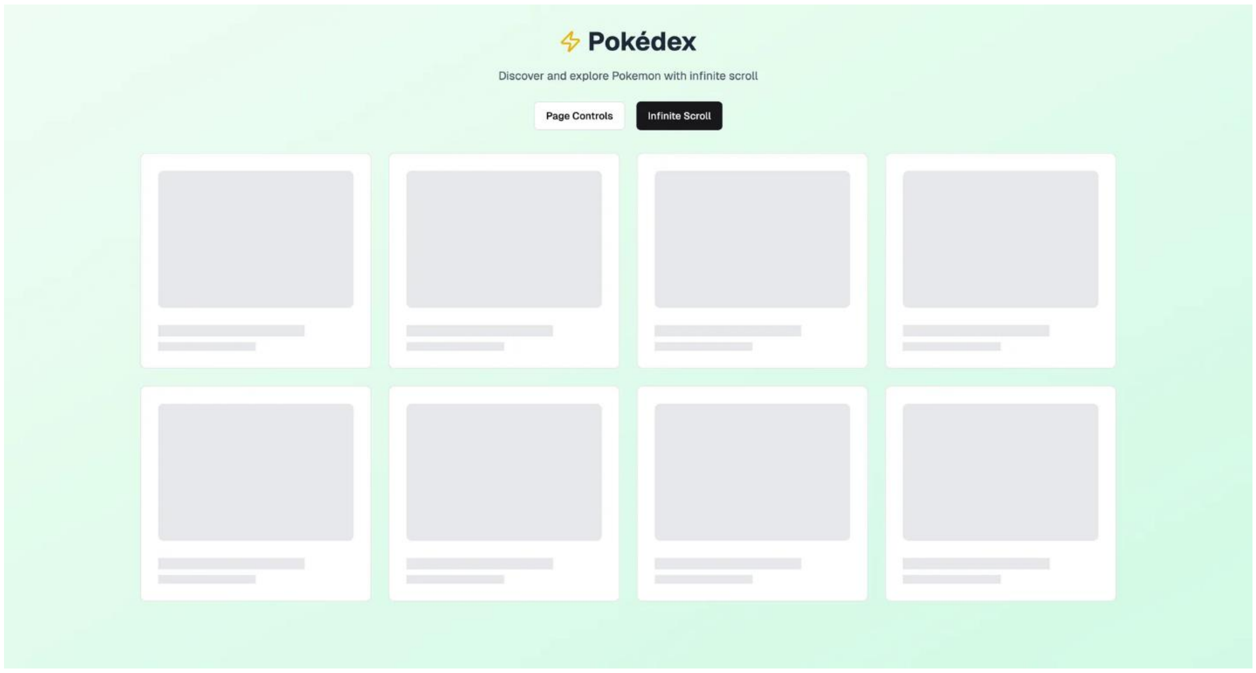
6. Git Usage

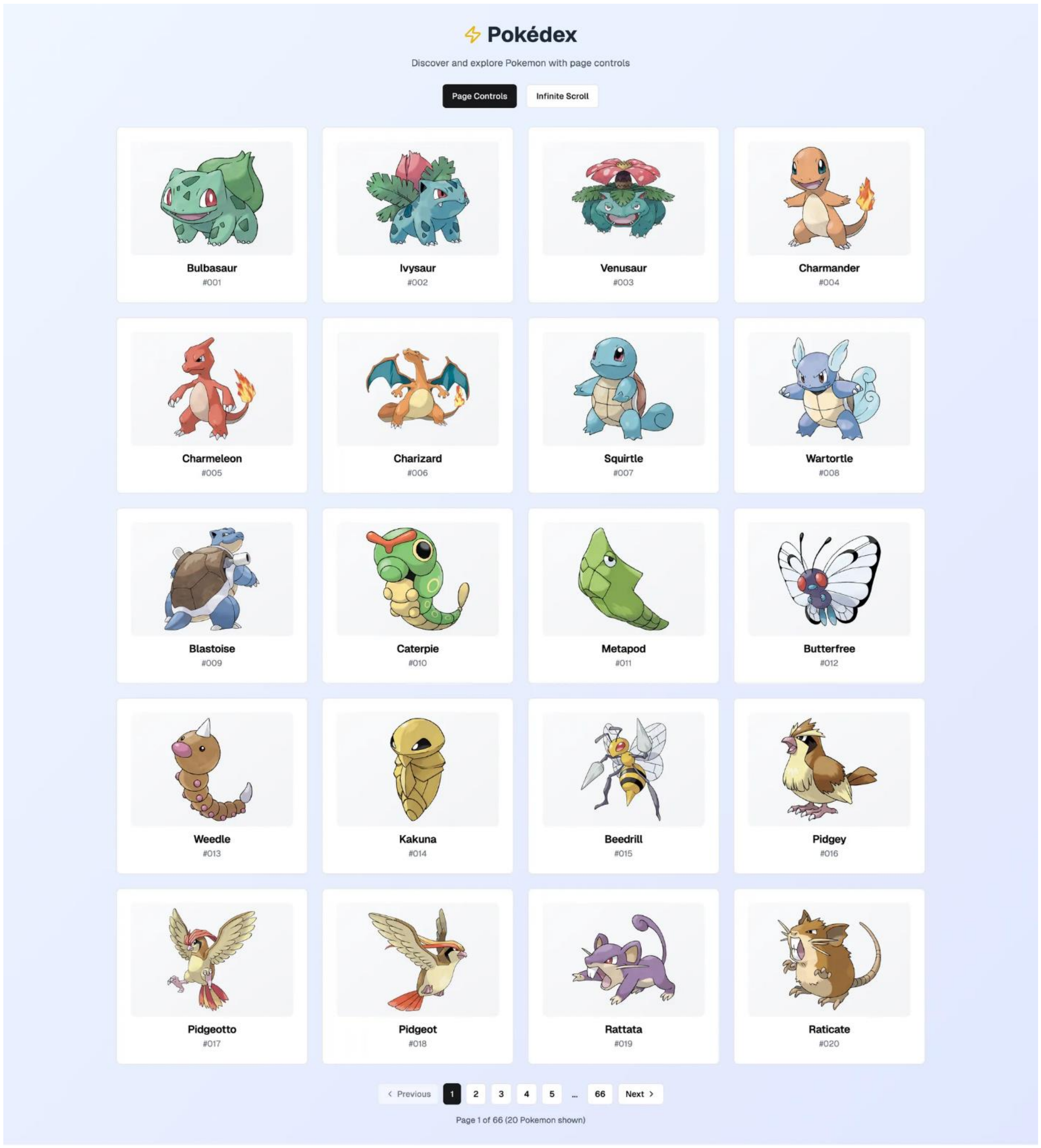
- Push your code to a **public Git repository**
- Use clear, meaningful commit messages that reflect progress and intent

7. Deployment

- Deploy the project using Vercel / Netlify / Cloudflare Pages
- Submit:
  - 🔗 **Live Preview URL**
  - 🔗 **GitHub Repository URL**

📸 Reference Designs



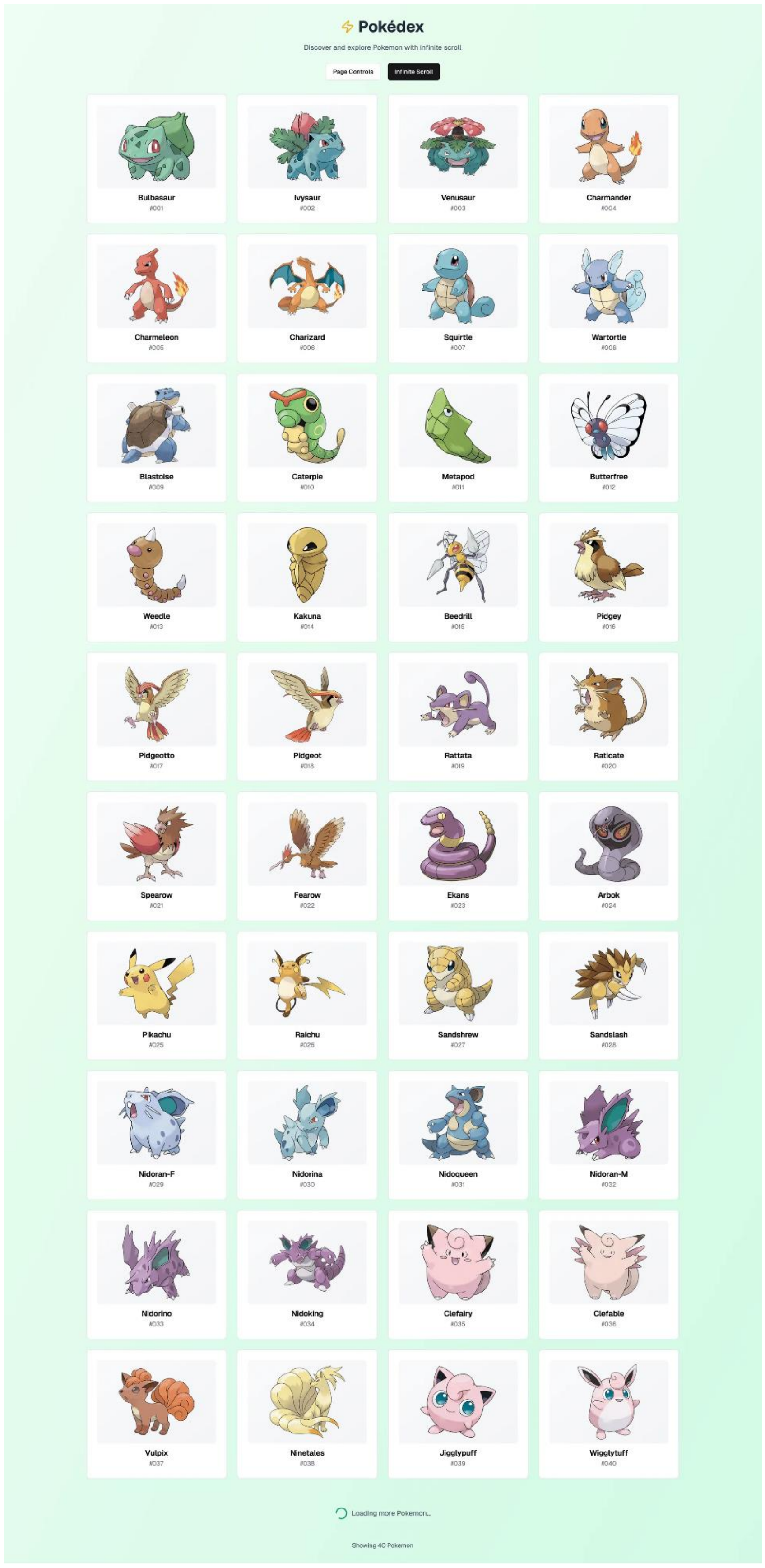


Pokédex

Discover and explore Pokemon with page controls

Bulbasaur #001
Ivysaur #002
Venusaur #003
Charmander #004
Charmeleon #005
Charizard #006
Squirtle #007
Wartortle #008
Blastoise #009
Caterpie #010
Metapod #011
Butterfree #012
Weedle #013
Kakuna #014
Beedrill #015
Pidgey #016
Pidgeotto #017
Pidgeot #018
Rattata #019
Raticate #020

Page 1 of 66 (20 Pokemon shown)



Pokédex

Discover and explore Pokemon with infinite scroll

Bulbasaur #001

Ivysaur #002

Venusaur #003

Charmander #004

Charmeleon #005

Charizard #006

Squirtle #007

Wartortle #008

Blastoise #009

Caterpie #010

Metapod #011

Butterfree #012

Weedle #013

Kakuna #014

Beedrill #015

Pidgey #016

Pidgeotto #017

Pidgeot #018

Rattata #019

Raticate #020

Spearow #021

Fearow #022

Ekans #023

Arbok #024

Pikachu #025

Raichu #026

Sandshrew #027

Sandslash #028

Nidoran-F #029

Nidorina #030

Nidoqueen #031

Nidoran-M #032

Nidorino #033

Nidoking #034

Clefairy #035

Clefable #036

Vulpix #037

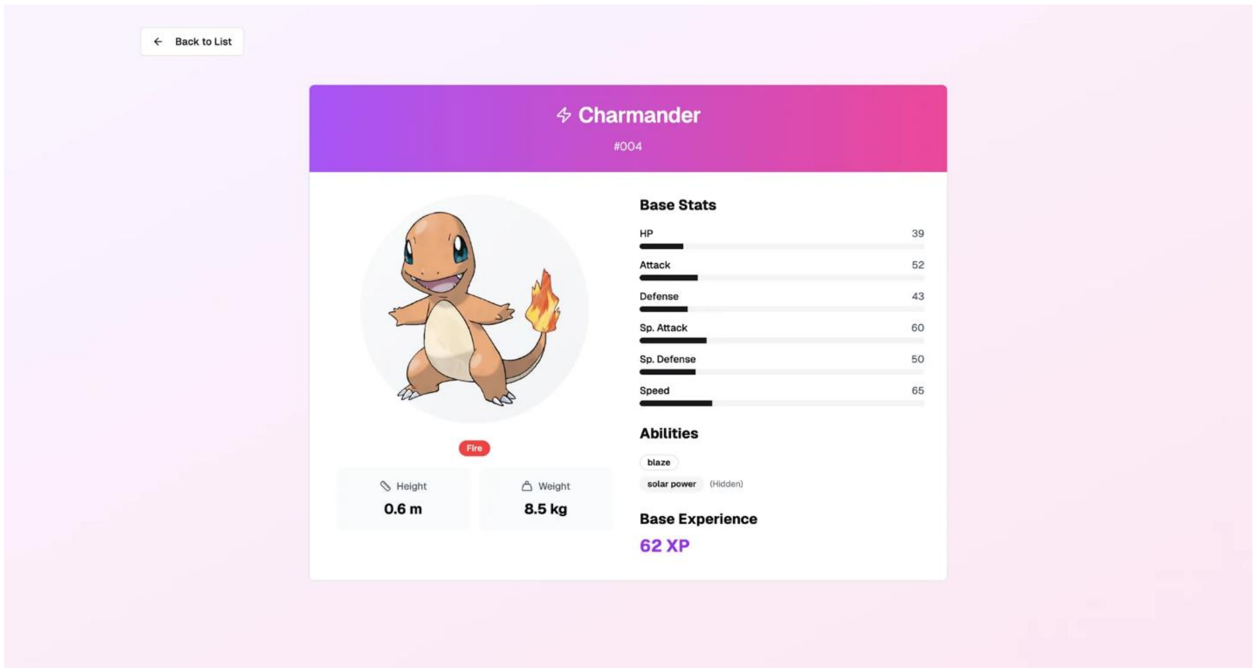
Ninetales #038

Jigglypuff #039

Wigglytuff #040

Loading more Pokemon...

Showing 40 Pokemon



Charmander

#004

Base Stats

HP	39
Attack	52
Defense	43
Sp. Attack	60
Sp. Defense	50
Speed	65

Abilities

blaze

solar power (Hidden)

Base Experience

62 XP

Fire

Height 0.6 m

Weight 8.5 kg

✴ Bonus (Optional)

Using any of the following will be considered a **plus**, but not required:

- [ ] **React Query** for API data fetching and caching
- [ ] **React Suspense** for managing loading statec s
- [ ] **Error Boundaries** for graceful runtime error handling
- [ ] **React Server Components (RSC)** if using a compatible setup

These demonstrate deeper understanding of scalable React architecture.

⏱ Timebox

- Complete the task within **4 hours**
- If incomplete, describe what you'd do next and submit what's finished

---

✅ Submission Checklist

- Pixel-perfect layout matching reference designs
- Fully responsive across desktop, tablet, and mobile
- Pagination and "Load More" views implemented
- Dedicated detail page functional and styled

- Loading and error states handled properly
- Code is modular and easy to test
- Publicly deployed with a working live link
- GitHub repo is public with meaningful commit history

🐻 Good Luck!

We're excited to see how you approach this challenge.

Focus on clean code, thoughtful structure, and user experience — and most importantly, have fun building! 🚀