🧑‍💻 Engineering Test

Welcome to the **Modinity Engineering Test**. We designed this challenge to evaluate your ability to **transform requirements into a working application**, not just your coding knowledge.

You are free to use **any language, framework, or stack** that you are most comfortable with. Please focus on **clarity, maintainability, and good engineering practices**.

📖 Story – Gotta Catch 'Em All!

At Modinity, we often integrate with **external APIs** to power our e-commerce and business applications. For this challenge, imagine that our company is launching a **Pokémon-themed merchandise store**. We want to provide customers with a simple **Pokémon Explorer** web app where they can:

1. **Browse Pokémon list** (fetched from PokeAPI)
2. **Search Pokémon by name**
3. **View Pokémon details** such as abilities, stats, and types
4. **Mark a Pokémon as a favorite** (stored in a simple DB)
5. **Build a team of up to 6 Pokémon** (stored in a simple DB)

🛠️ Requirements

Your application must include the following:

1. **Pokémon List Page**
   - Fetch Pokémon list with pagination (use PokeAPI endpoints).
   - Show Pokémon image, name, and basic type(s).
2. **Search**
   - Search Pokémon by name.
   - Display results in the same list format.
3. **Pokémon Detail Page**
   - Show name, image, abilities, stats, and types.
   - Display at least **three key stats visually** (e.g., HP, Attack, Defense as a bar chart or similar).
4. **Favorites**
   - Allow marking/unmarking Pokémon as favorites.

  - Show a **Favorites tab/page** where users can view their saved Pokémon.
  - **Favorites must be persisted** in a lightweight database (e.g., SQLite or JSON file) through a backend API.
5. **Type Filter (Challenge)**
   - Add a **filter by type** (e.g., Fire, Water, Grass).
   - The list should update based on selected type(s).
6. **Simple Team Builder**
   - Allow the user to create a **team of up to 6 Pokémon**.
   - Display the team on a separate page.
   - Prevent duplicates in the same team.
   - **Team data must be persisted** in a lightweight database (e.g., SQLite or JSON file) through a backend API.
7. **Persistence (Required)**
   - Implement a **backend service**.
   - Backend must handle CRUD operations for:
     - Favorites
     - Team Pokémon
   - Use a **simple database** (SQLite recommended, JSON file acceptable).
   - Document your backend API endpoints (README or OpenAPI/Swagger if possible).

🚀 Bonus (Optional, Extra Points)

- Deploy the application and share the **live URL** with us (Netlify, Vercel, Heroku, Render, GCP, AWS, etc.).
- If deployment is not possible, include a **short video recording** in your PR showing:
  - How the app runs
  - The steps to deploy
- Add a **basic test suite** (unit/integration tests).

📁 How to Submit

1. Create a new **Repository** on your GitHub account.
2. Upload this PDF to the `docs` directory and create a branch with your preferred name.
3. Add a short **README.md** inside your project folder explaining:
   - Tech stack used
   - How to run locally
   - Deployment steps (if any)

4. After you have finished with your code, open a **Pull Request (PR)** to this repository and **assign it to our GitHub account** (`modinity-tech`).
5. **Notify us by email** once the PR is created: send an email to `tech@modinity.com` with:
    - Subject: `Engineering Test Submission – <Your Name>`
    - Content: PR link, repository name, branch name, and (optional) live demo/video links.

🙌 Final Notes

This test is not about who can write the most code.
It's about showing us how you:

- Understand requirements
- Structure and deliver an application
- Use engineering best practices
- Communicate through code and documentation
- Handle both **frontend and backend responsibilities**

👉 Please don't over-engineer – focus on delivering a clean and functional solution.

Good luck, and may the best trainer win! 🐉 ✨
