Frontend Developer Exercise:   
Carbon Emissions Dashboard

Hello candidate!

Thank you for your interest in HanaLoop, a startup that is continuously growing. Your interest in our company shows that you are aware of the impact of climate change. At HanaLoop we are building a platform to help organizations to help with their journey to Net Zero.

We are looking for new team members that are passionate about solving problems which requires constant learning.

We would like you to show off your skills and knowledge in front-end development with this assignment.

## **Purpose**

The purpose of this assignment is to evaluate your ability to design and build a small, production-minded **web application** that applies good practices, and demonstrates good product/design judgment. We’re looking for thoughtful tradeoffs, not just feature-count.

This is your opportunity to impress!

## What you are building

You will be building a **basic web-based Carbon (aka Greenhouse Gas) Emissions Dashboard**. The app should include a navigation drawer and a main content area with visualization of emissions data.

The dashboard is for executives and managers to understand the emission of thor companies and affiliates. So they can plan ahead the carbon taxes they need to pay.

You are expected to use the data model (but not limited to) as specified in the Data Model section below to fill the content area. You can be as creative as possible to produce an engaging Dashboard.

## **What we’re assessing**

1. **Creativity and Critical thinking**: ability to spot ambiguity, ask questions, document assumptions, and propose sensible extensions, come up with interesting new ideas.
2. **Sound UI/UX Design, aesthetics**: A modern look and feel with an intuitive interface, consistent use of UI components, and a well-balanced color palette.
3. **UI engineering**: Responsive layout, loading/error states. Clean separation layout/UI state, filter state, and data state.
4. **Software engineering**: Modularity, performance, .
5. **Code quality**: Readability and testability. Proper use of types, structure, decomposition, naming, tests where it counts, and a readable commit history.

The emphasis is not only on building the features, but on demonstrating creativity and thoughtful design choices.

## **As you work on the assignment**:

Document your assumptions and key design decision if applicable. We care about your reasoning. Also time it took to complete the assignment.

Feel free to send us clarifying questions for clarifications.

## **Timebox**

**T**his assignment was designed to fit in 8 –12 hours of focused work. As you can use resources on the internet. But of course, the overall design must be your own creation.

Do not exceed **3 days** in total.

## **Technical** C**onstraints**

1. **Next.js 14+ (App Router) + React 18 + TypeScript**.
2. No heavy component libraries (e.g., MUI/Ant) for the main UI. Small utilities (headless UI/shadcn) are permitted.
3. Styling: any approach is fine (CSS Modules, Tailwind). Show a basic design system.
4. You may use a state lib (e.g., Zustand/Redux) or React Context + custom hooks.

## Data model

Below are the data model that you can use for the Dashboard:

The data you will be working with (countries, companies, emissions, and posts) mimics what you might encounter in a real-world application.

### **Companies (with monthly emissions)**

| type Company = {  id: string;  name: string;  country: string; // Country.code  emissions: GhgEmission[]; }; |
| --- |

### Emission

| type GhgEmission = {  yearMonth: string; // "2025-01", "2025-02", "2025-03"  source: string; // gasoline, lpg, diesel, etc  emissions: number; // tons of CO2 equivalent }; |
| --- |

### **Posts (linked to company + month)**

| type Post = {  id: string;  title: string;  resourceUid: string; // Company.id  dateTime: string; // e.g., "2024-02"  content: string; }; |
| --- |

## Seed data example

| type Post = {  id: string;  title: string;  resourceUid: string; // Company.id  dateTime: string; // e.g., "2024-02"  content: string; }; |
| --- |

| export const companies: Company[] = [  {  id: "c1",  name: "Acme Corp",  country: "US",  emissions: [{ "yearMonth": "2024-01", "emissions": 120}, { "yearMonth": "2024-02": "emissions": 110}, {"yearMonth": "2024-03": "emissions": 95 }]  },  {  id: "c2",  name: "Globex",  country: "DE",  emissions: [{ "yearMonth": "2024-01", "emissions": 80}, { "yearMonth": "2024-02": "emissions": 105}, {"yearMonth": "2024-03": "emissions": 120 }]  } ];  export const posts: Post[] = [  {  id: "p1",  title: "Sustainability Report",  resourceUid: "c1",  dateTime: "2024-02",  content: "Quarterly CO2 update"  } ]; |
| --- |

## **Fake backend (provided stub)**

Create a module (e.g., lib/api.ts) that simulates network I/O, latency (200–800ms), and occasional failure (10–20%) for write operations.

| // lib/api.ts let \_countries = [...countries]; let \_companies = [...companies]; let \_posts = [...posts];  const delay = (ms: number) => new Promise(res => setTimeout(res, ms)); const jitter = () => 200 + Math.random() \* 600; const maybeFail = () => Math.random() < 0.15;  export async function fetchCountries() {  await delay(jitter());  return \_countries; }  export async function fetchCompanies() {  await delay(jitter());  return \_companies; }  export async function fetchPosts() {  await delay(jitter());  return \_posts; }  export async function createOrUpdatePost(p: Omit<Post, "id"> & { id?: string }) {  await delay(jitter());  if (maybeFail()) throw new Error("Save failed");  if (p.id) {  \_posts = \_posts.map(x => x.id === p.id ? (p as Post) : x);  return p as Post;  }  const created = { ...p, id: crypto.randomUUID() };  \_posts = [...\_posts, created];  return created; } |
| --- |

*This lets you test loading, error, partial failure, and* ***rollback*** *behavior realistically.*

## **Deliverables**

As a modern developer, you would work on a git repo. You can use GitHub or Gitlab.

Once you are completed, share it with us.

At minimal, the repo should

1. **Have a runnable** Next.js app.

2. Include a **README** with brief intro and how to test/run it:

You can add additional documentation, for example

* Questions you asked or assumptions you made (top of file).
* Architecture overview (state boundaries, data flow diagram or bullets).
* Rendering efficiency notes (what rerenders when, and why).
* Any tradeoffs or shortcuts due to time.
* Design rationale, a brief write-up explaining key UI decisions (layout, motion etc).

## **Evaluation rubric**

1. **Creativity and Critical thinking**: 25 %
2. **Sound UI/UX Design, aesthetics**: 25%
3. **UI engineering**: 20%.
4. **Software engineering**: 20%
5. **Code quality**: 10%