Understanding Our Weather App: A TypeScript Journey 🌤️

Table of Contents

Introduction

Hey there! 👋 This guide will help you understand how our weather app works. Imagine you're building a LEGO set - we'll look at each piece, understand what it does, and see how they all fit together to create something amazing!

Our weather app can:

- Show weather for Swedish cities 🏰
- Tell you today's temperature and forecast 🌡️
- Show weather for the next few days ⏳
- Let you use your location to get local weather 📍
- Show you when the sun rises and sets 🌅

Basic Building Blocks

1. Types and Cities

```
type City = { name: string; lat: number; lon: number };
const cities: City[] = [
  { name: 'Stockholm', lat: 59.3293, lon: 18.0686 },
  { name: 'Malmö', lat: 55.605, lon: 13.0038 },
  { name: 'Gothenburg', lat: 57.7089, lon: 11.9746 }
];
```

Think of a `City` as a special label that says "anything with this label must have a name and location (latitude and longitude)". It's like a form that must be filled out with:

- A name (like "Stockholm")
- A latitude number (like 59.3293)
- A longitude number (like 18.0686)

We create an array (list) of cities, each following this pattern. This helps TypeScript make sure we don't accidentally put wrong information in our cities!

2. Weather Icons and Descriptions

```
const weatherIcons: { [key: number]: string } = {
  1: 'icons/day/01.svg',
  2: 'icons/day/02.svg',
  // ... more icons
};

const weatherText: { [key: number]: string } = {
  1: 'Clear',
  2: 'Nearly clear',
  // ... more descriptions
};
```

These are like translation dictionaries:

- `weatherIcons`: Converts weather codes (numbers) into picture files
- `weatherText`: Converts those same numbers into words we can read

For example:

- If the weather code is 1:
  - `weatherIcons[1]` gives us a sunny icon (`01.svg`)
  - `weatherText[1]` tells us it's "Clear"

3. Helper Functions

```
function qs<T extends HTMLElement>(sel: string) {
  return document.querySelector(sel) as T | null;
}

function mapSymbolToIcon(sym: number) {
  return weatherIcons[sym] || 'icons/day/01.svg';
}
```

These are our tools that make common tasks easier:

- `qs`: Finds elements on our webpage (like finding a specific LEGO piece in a big box)
- `mapSymbolToIcon`: Gets the right weather icon (if it can't find one, it uses a default sunny icon)

Working with Weather APIs

1. The SMHI Weather API

```
const SMHI_POINT = (lat: number, lon: number) =>
  `https://opendata-download-
```

```
metfcst.smhi.se/api/category/pmp3g/version/2/geotype/point/lon/${lon}/lat/${lat}/data.json`;
```

This creates the web address where we get our weather data. It's like knowing exactly which shelf in a library has the book you want! We just need to tell it where we are (latitude and longitude).

2. Fetching Weather Data

```
async function fetchPointData(lat: number, lon: number) {
  const url = SMHI_POINT(lat, lon);
  const res = await fetch(url);
  if (!res.ok) throw new Error('Fetch error ' + res.status);
  return res.json();
}
```

This function:

1. Takes a location (lat/lon)
2. Asks SMHI's servers for weather data
3. Waits for the answer (`await`)
4. Converts the answer from computer-language to JavaScript objects

It's like sending a letter to SMHI saying "What's the weather at these coordinates?" and waiting for their reply!

3. Organizing Weather Data

```
function groupByDate(timeSeries: any[]) {
  const days: { [date: string]: any[] } = {};
  timeSeries.forEach(entry => {
    const date = entry.validTime.split('T')[0];
    if (!days[date]) days[date] = [];
    days[date].push(entry);
  });
  return days;
}
```

This function organizes weather data by date. Imagine having a stack of weather reports and sorting them into different folders, one for each day!

Making the Page Come Alive

1. Rendering a City's Weather

```
async function renderCity(city: City) {
  try {
    const data = await fetchPointData(city.lat, city.lon);
```

```
    const series = data.timeSeries as any[];
    // ... more code
  } catch (err) {
    console.error('renderCity error', err);
  }
}
```

This is our main function that:

1. Gets weather data for a city
2. Updates all the parts of our webpage with new information
3. Handles any errors that might happen

It's like a conductor in an orchestra, making sure every part plays its role perfectly!

2. Showing Current Weather

```
const current = series[0];
const t = current.parameters.find((p: any) => p.name === 't')?.values[0];
const sym = current.parameters.find((p: any) => p.name === 'Wsymb2')?.values[0];
```

This part:

- Gets the current temperature (`t`)
- Gets the weather symbol (`Wsymb2`)
- Updates the big display at the top of our page

3. Creating Weather Cards

```
dates.forEach((date, idx) => {
  // ... card creation code
  card.innerHTML = `
    <p class="weak-day">${dayName}</p>
    <div class="weak-two">
      <span class="weak-icon"><img src="${mapSymbolToIcon(sym)}"
alt="${weatherText[sym] || ''}" width="28"/></span>
      <p class="weak-weather-info">${weatherText[sym] || String(sym)}</p>
    </div>
    <p class="weak-degree">${Math.round(min)}° / ${Math.round(max)}°</p>
  `;
});
```

For each day, we create a card showing:

- The day name
- A weather icon
- A description of the weather

- The highest and lowest temperatures

Advanced Features

1. Sunrise and Sunset Times

```
const srUrl = `https://api.sunrise-sunset.org/json?
lat=${city.lat}&lng=${city.lon}&formatted=0`;
```

We use a different API to get sunrise and sunset times. This shows when the sun will rise and set in your city!

2. Geolocation

```
navigator.geolocation.getCurrentPosition((pos) => {
  const lat = pos.coords.latitude;
  const lon = pos.coords.longitude;
  const userCity = { name: 'Your location', lat, lon };
  renderCity(userCity);
});
```

When you click the map icon:

1. Asks your browser "Where am I?"
2. Browser asks your permission
3. If you agree, gets your location
4. Shows weather for where you are!

Putting it All Together

Our app works like this:

1. When you open it:
    - Shows Stockholm's weather by default
    - Gets sunrise/sunset times
    - Shows today's forecast by hours
    - Shows next 4 days' forecast
2. When you search or use geolocation:
    - Gets new weather data
    - Updates all the displays
    - Shows new sunrise/sunset times
3. For each weather update:
    - Converts weather codes to icons and text
    - Shows temperatures and conditions

    - Updates all the cards and details

Cool Technical Details 🤓

1. **Error Handling**: We use `try/catch` blocks to handle problems gracefully - if something goes wrong, the app won't crash!
2. **Type Safety**: TypeScript helps us avoid mistakes by checking our data types - it's like having a helpful friend looking over our shoulder saying "Are you sure about that?"
3. **Smart Defaults**: If something's missing (like a weather icon), we have backup plans (default sunny icon)!
4. **Clean Code**: Our functions each do one specific job and do it well - like having specialized tools instead of one complicated multi-tool.

Want to Learn More? 🎓

Try these exercises:

1. Look at the SMHI API documentation to see what other weather data we could show
2. Try adding more cities to the `cities` array
3. Experiment with different time formats for sunrise/sunset
4. Add more weather parameters to the details section

Remember: The best way to learn is to experiment! Try changing things and see what happens - just keep a backup of the working code first! 😉

Final Tips 💡

1. Always check the browser console (F12) if something's not working
2. Use the Network tab to see API calls in action
3. TypeScript errors are your friends - they prevent bugs!
4. Comments in the code explain the "why" behind the "what"

Happy coding! 🚀