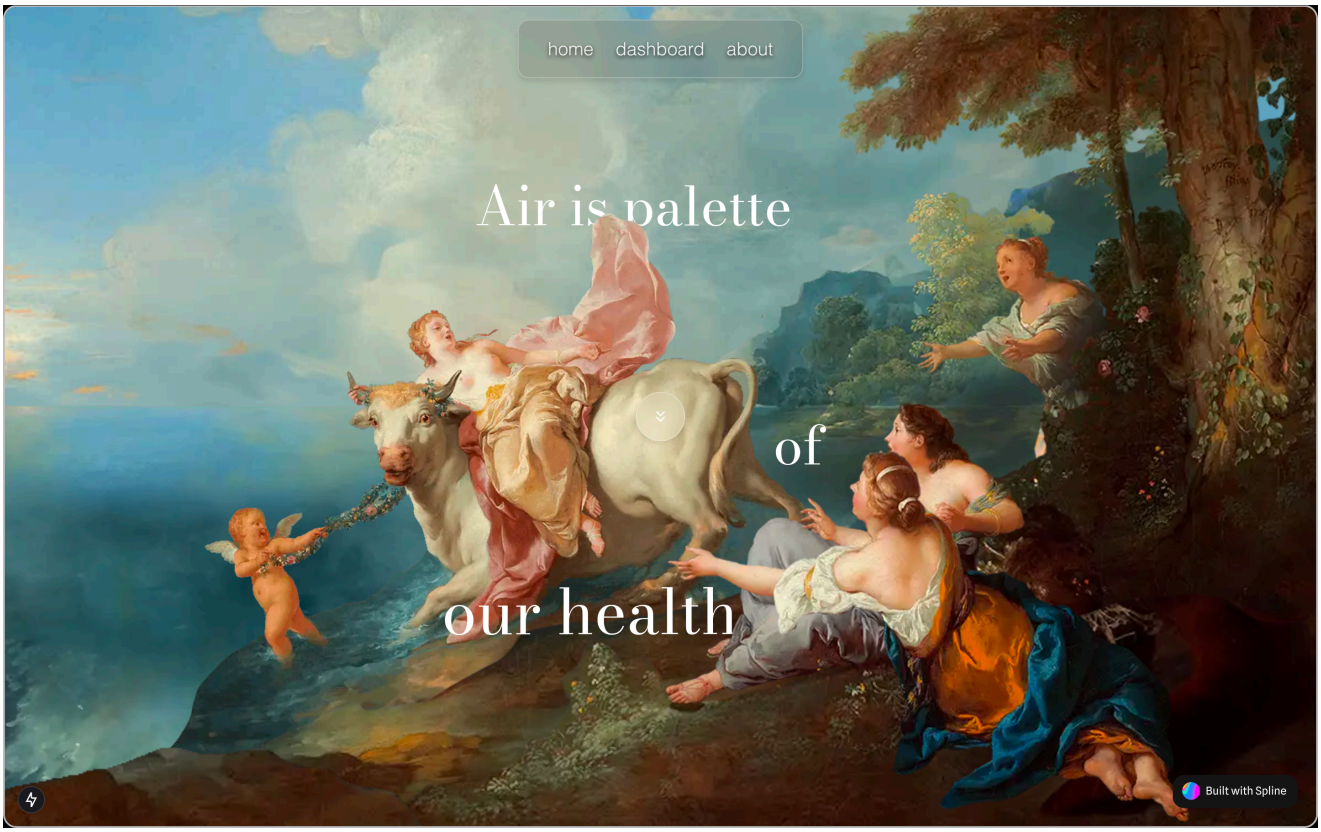
WEB APPLICATION PROGRAMMING

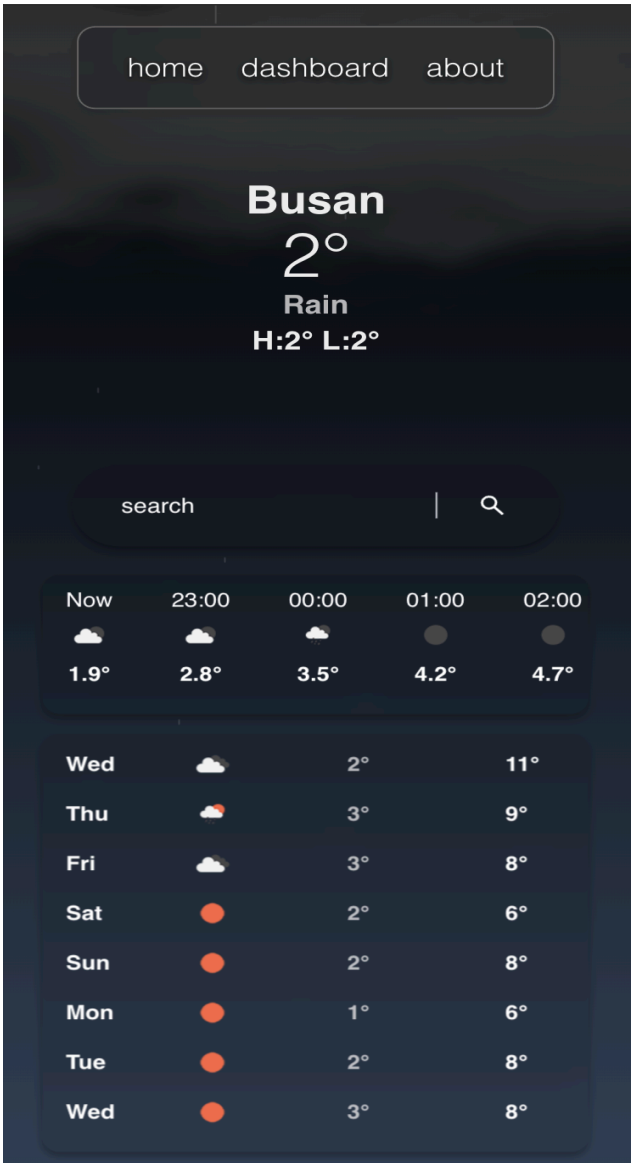


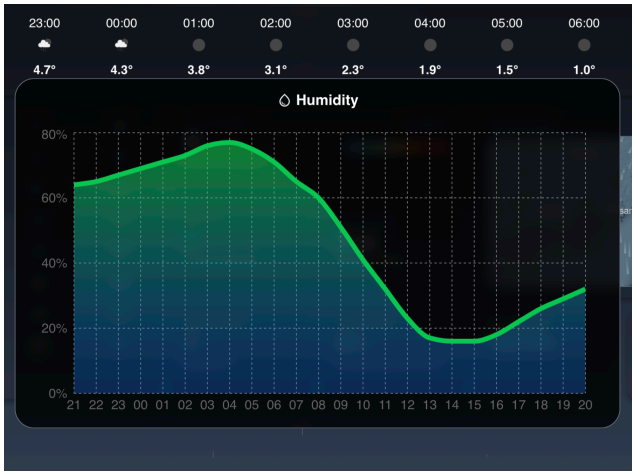
Outline

This report explores on the Web Application Programming 2024 final project. It consists of project screenshots, summary, project structural components, APIs and libraries utilized, and finally - successes, failures, and what can be improved sections.

- github link
- project demo link

Screenshots





Project Summary

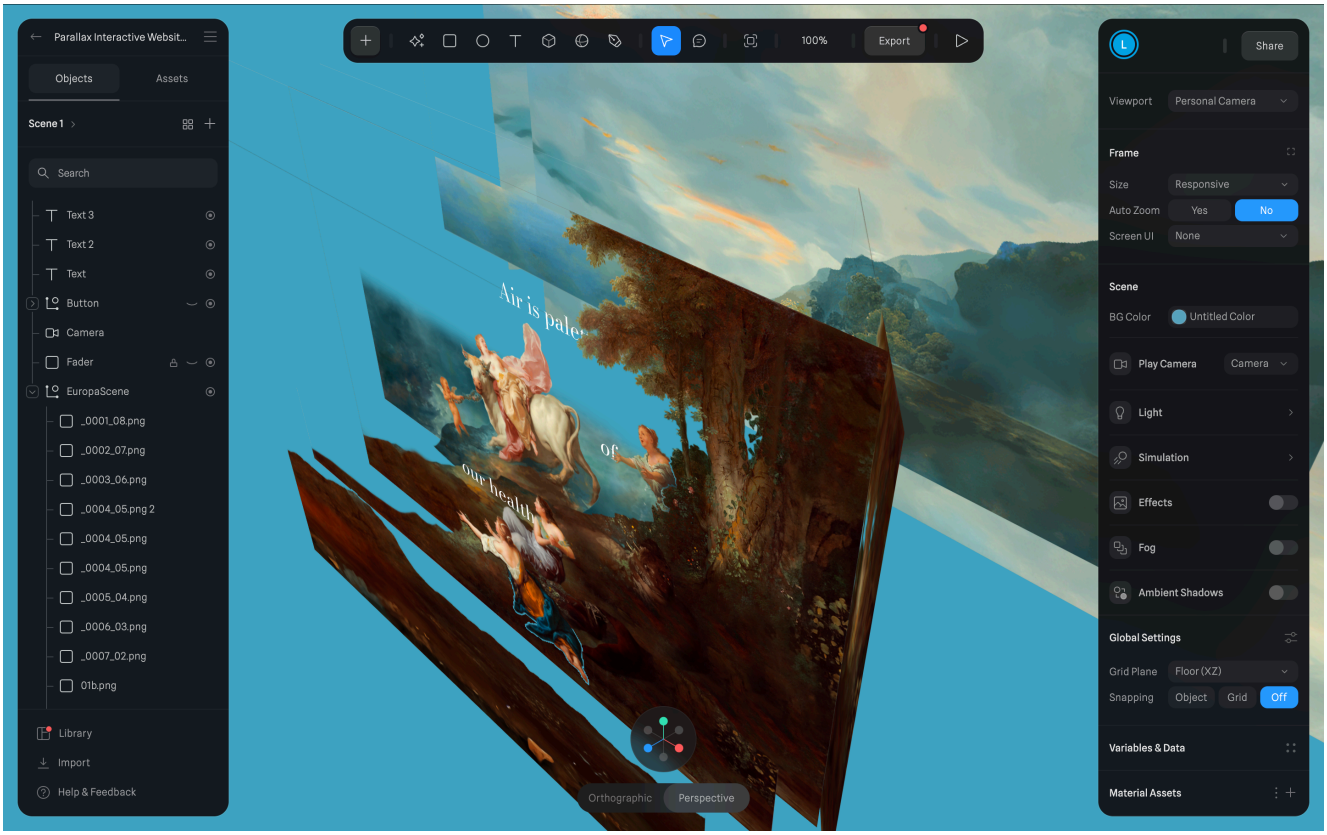
Main Page serves as a cover for the functional part - Dashboard - and displays background information about air pollutants, importance of living in clean air environment, footer with credits, and links to my social media, and design solutions to capture user attention.

Dashboard is the heart of the project and contains a grid consisting of air and weather indicators. Background color and animations change depending on the weather conditions displayed. Grid elements can be tapped to show an overlay, that shows detailed information concerning the tapped indicator.

About Page features a grid with frequently asked questions and answers to them. It displays what was used to create this website and explain purpose of use for each component.

Main Page

Spline Container

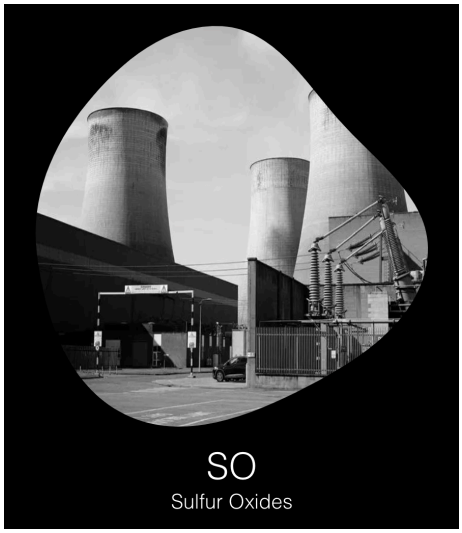


Spline Container is a 3D scene that is shown when user first enters the website. Spline is a web tool for creating animated scenes featuring 3D objects. For the project, I used Spline to create a multi-layered picture that reacts to mouse cursor location. The key is a different angle of turning that creates a smooth parallax effect.

Floating Blobs

Floating Blobs (freeform, abstract shapes that have smooth, curved edges) are realized using <svg> with transparent inside and black outside. On top of it is placed and image. CSS is used to animate different shapes and transitions.

```
<div className="cont">
    <svg className="hehe" width="1000" height="1000" viewBox="0 0 1000 1000" fill="none"
        <path className="pathThree" fillRule="evenodd" clipRule="evenodd" d="M0 0H1000V1
    </svg>
    <Image
        src={nnn}
        alt="asd"
        className="imaege"
    />
</div>
<p className="text-center font-thin text-4xl">SO</p>
<p className="text-center font-thin text-xl">Sulfur Oxides</p>
```

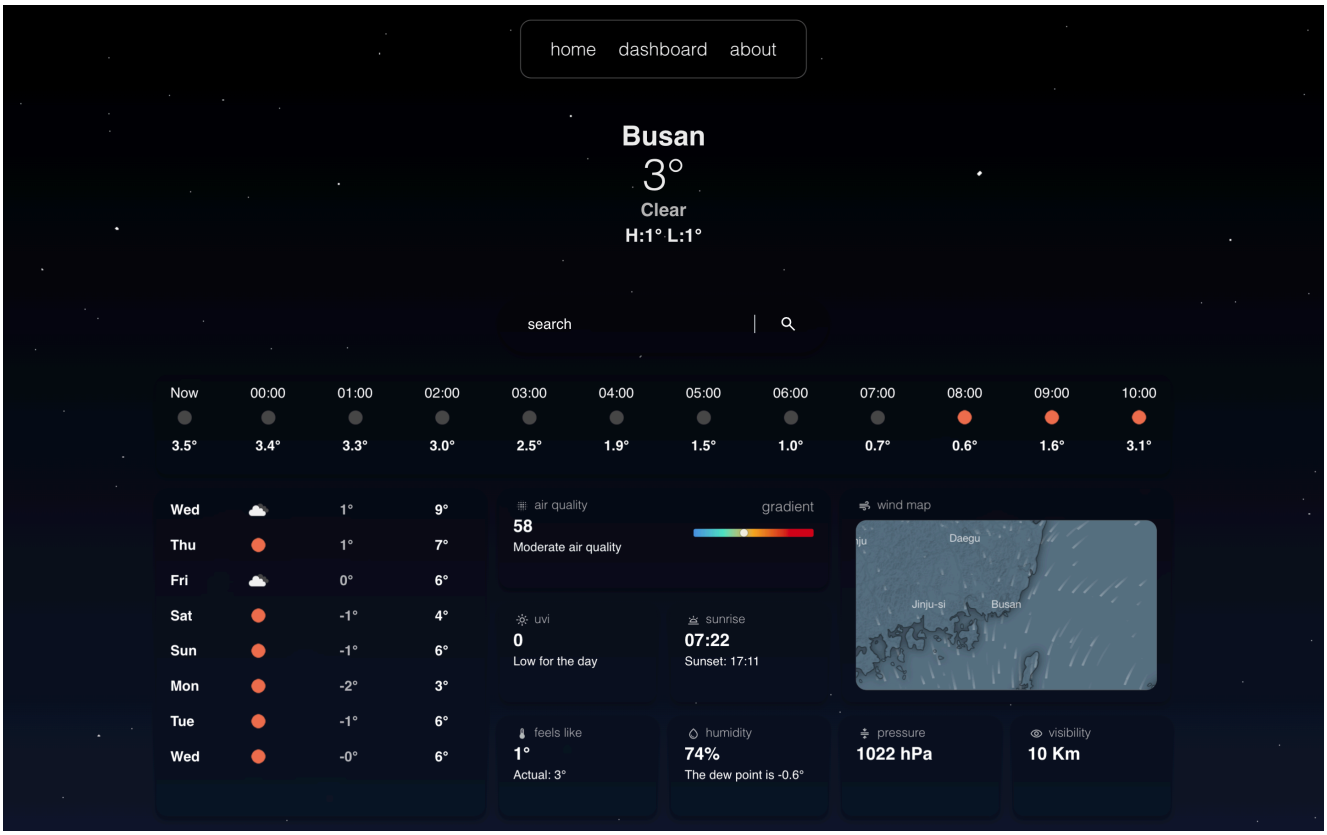


Parallax Section

```
<motion.div style={{ y: yOffset }} className="absolute flex flex-row min-w-full ju
        <Image
            src={tt}
            alt="asd"
            className="md:w-[40vh] md:h-[40vh] object-fill"
        />
</motion.div>
<div className="hidden md:block">
    <motion.div style={{ y: yOffsetTwo }} className="absolute left-40 bottom-0">
        <Image
            src={dd}
            alt="asd"
            className="w-[30vw] h-[30vh] object-fit"
        />
    </motion.div>
</div>
```

Motion Library and Tailwind were used to create design and offset between image positions to create parallax effect.

Dashboard


home dashboard about
Busan
3°
Clear
H:1° L:1°
search
Now 00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00 08:00 09:00 10:00
3.5° 3.4° 3.3° 3.0° 2.5° 1.9° 1.5° 1.0° 0.7° 0.6° 1.6° 3.1°
Wed 1° 9°
Thu 1° 7°
Fri 0° 6°
Sat -1° 4°
Sun -1° 6°
Mon -2° 3°
Tue -1° 6°
Wed -0° 6°
air quality
58
Moderate air quality
gradient
wind map
Daegu
Jinju-si
Busan
uvi
0
Low for the day
sunrise
07:22
Sunset: 17:11
feels like
1°
Actual: 3°
humidity
74%
The dew point is -0.6°
pressure
1022 hPa
visibility
10 Km


home dashboard
Madrid
7°
Clouds
H:5° L:5°
madrid
Now 21:00 22:00 23:00 00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00
7.2° 7.2° 7.4° 7.7° 8.1° 8.2° 7.7° 7.5° 7.3° 6.9° 6.4° 5.6°
Wed 5° 8°
Thu 5° 9°
Fri 5° 9°
Sat 4° 9°
Sun 6° 11°
Mon 4° 10°
Tue 6° 10°
Wed 5° 10°
air quality
79
Good air quality
gradient
Sigüenza
Ávila
Madrid
SPAIN
Cuenca
Toledo
uvi
0
Low for the day
sunrise
16:28
Sunset: 01:48
feels like
6°
Actual: 7°
humidity
56%
The dew point is -0.8°
pressure
1019 hPa
visibility
10 Km

```
<div className="grid grid-cols-6 grid-rows-13 md:grid-rows-5 gap-4 md:h-[70vh] w-[90vw] md:w-[80vw]
    <HourlyForecast weatherData={weatherData} onClick={handleHourlyForecastClick}/>
    <DailyForecast weatherData={weatherData} onClick={handleDailyForecastClick}/>
    <AirQuality airQualityData={airQualityData} onClick={handleAirQualityClick}/>
    <WindMap weatherData={weatherData} lat={lat} lon={lon}/>
    <UVIndex weatherData={weatherData} uviMessage={uviMessage} onClick={handleUVForecastClick}/>
    <Sunrise weatherData={weatherData} onClick={handleSunriseClick}/>
    <FeelsLike weatherData={weatherData} onClick={handleFeelsLikeClick}/>
    <Humidity weatherData={weatherData} onClick={handleHumidityClick}/>
    <Pressure weatherData={weatherData} onClick={handlePressureClick}/>
    <Visibility weatherData={weatherData} onClick={handleVisibilityClick}/>
</div>
```

Dashboard is a page consisting of three components: city, search, and grid. The screenshot displays a structure of the grid. Each subcomponent gets the data from fetched json. Clicking is handled using useEffect hook, a key feature in React that allows developers to perform side effects in functional components. It replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount in class components.

```
export const fetchWeatherData = async (latitude, longitude) => {
    try {
        const response = await fetch(
            `https://api.openweathermap.org/data/3.0/onecall?lat=${latitude}&lon=${longitude}&appid
        );
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error fetching weather data:', error);
    }
};
```

The screenshot above features API fetch function. Latitude and longitude are obtained using “navigator.geolocation” API, part of the HTML5 Geolocation API, native to most browsers..

```
useEffect(() => {
    const getCurrentLocation = async () => {
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(async (position) => {
                const { latitude, longitude } = position.coords;
                setLocation({ latitude, longitude });
                try {
                    const airQualityData = await fetchAirQualityData(latitude, longitude);
                    setAirQualityData(airQualityData);
                    const weatherData = await fetchWeatherData(latitude, longitude);
                    setWeatherData(weatherData);
                    const cityData = await fetchCityData(latitude, longitude);
```

Below is a list of crucial variables that keep track of values used in the page.

```
const [isMobile, setIsMobile] = useState(false);
const [airQualityData, setAirQualityData] = useState(null);
const [weatherData, setWeatherData] = useState(null);
const [location, setLocation] = useState({ latitude: null, longitude: null });
const [mapLocation, setMapLocation] = useState({ lat: 35.1731, lon: 129.0714 });
const [city, setCity] = useState('');
const [scrolled, setScrolled] = useState(false);
const insideRef
const { scrollY
import useTransform
ner: insideRef });
const yOffset = useTransform(scrollY, [-100, 100], isMobile ? [0, 0] : [0, -100]);
const yOffsetCity = useTransform(scrollY, [-100, 100], isMobile ? [0, 0] : [0, 30]);
const [searchCity, setSearchCity] = useState('');
const now = new Date();
const startOfToday = new Date(now.setHours(0, 0, 0, 0)).getTime() / 1000;
const endOfToday = startOfToday + 86400;
const todayData = weatherData?.hourly.filter(
    (entry) => entry.dt >= startOfToday && entry.dt < endOfToday
);
const maxUVI = todayData?.reduce((max, entry) => Math.max(max, entry.uvi), 0);
const uviRanges = [
    { range: [0, 2], message: "Low for the day" },
    { range: [3, 5], message: "Moderate for the day" },
    { range: [6, 7], message: "Will be high today" },
    { range: [8, 10], message: "Extreme, avoid sun" },
    { range: [11, Infinity], message: "UV index is Extreme. Avoid the sun entirely." },
];
const uviMessage =
    uviRanges.find(({ range }) => maxUVI >= range[0] && maxUVI <= range[1])?.message ||
    "No UV data available.";
```

Page dynamically switches background and THREE.js animation depending on current weather summary.

```
switch(weatherIcon) {
    case '01d':
    case '01n':
        return <NightStars />;
    case '03d':
        return <Clouds />;
    case '03n':
        return <Clouds />;
```

APIs and Libraries

`GOOGLE AIR QUALITY API:`

- is used to fetch air quality data. I preferred google api because it provides more detailed data about the air pollutants compared to openweather.

`OPENWEATHER API:`

- is used to fetch primarily weather data. It is used for most components like humidity, visibility, pressure, etc.

`WINDY API:`

- is used to display wind map for the dashboard. It generates a map for the given location and wind direction animation..

`NOMINATIM API:`

- is used to fetch city given the location and vice versa, fetch desired city's latitude and longitude given the city name. For example, I used nominatim to get current location's city name like Busan. Then, I used this api to get a searched city's latitude and latitude to fetch this location using previous APIs.

`MOTION:`

- is used to create simple and lightweight animations for grid elements. Using it, I did tap and hover animations.

`SPLINE:`

- is used for creating a spline scene discussed in Main Page section..

`TAILWIND CSS:`

- is used to design CSS styles. It is quite similar to Bootstrap and I utilized it out of personal preference.

`RECHARTS:`

- is used to create graphs and charts for the fetched data.

`D3:`

- is used to create air quality index gradient. Universal AQ index is commonly interpreted on a gradient scale where blue represent excellent air quality and red is used to indicate poor air quality. I tried to replicate such gradient.

`THREE.JS:`

- is used to create cloud and rain animations. Below is an example code for creating THREE.js scene.

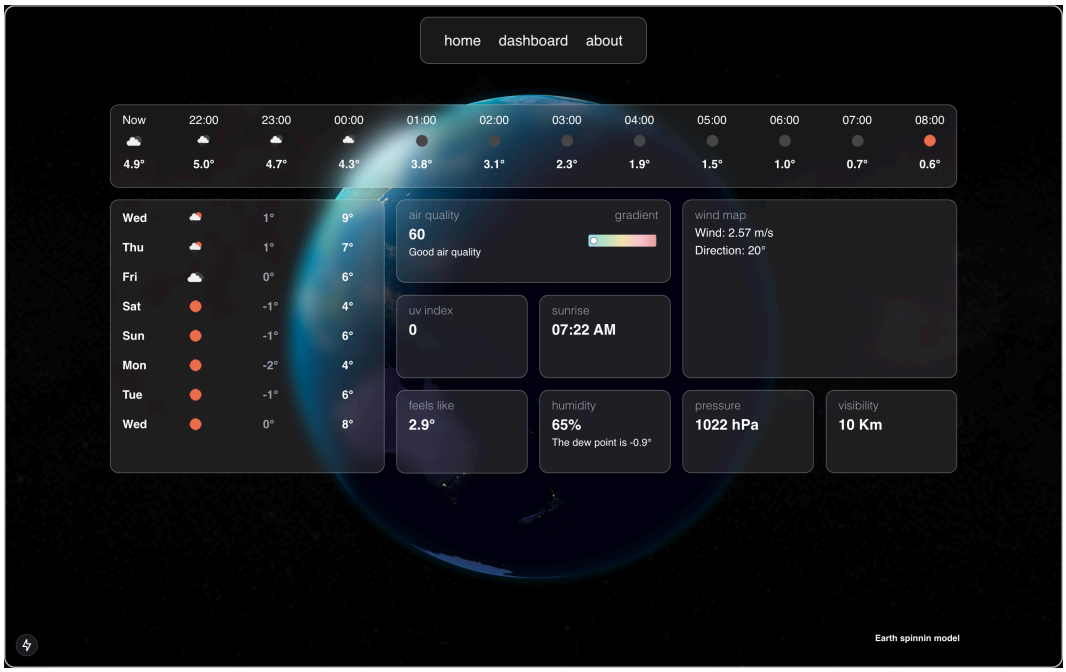
```
// Add Lighting
const ambientLight = new THREE.AmbientLight(0xffffff, 0.7);
scene.add(ambientLight);

const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.position.set(5, 10, 7.5);
scene.add(directionalLight);

// Add Clouds (First Cloud)
const cloudTexture1 = new THREE.TextureLoader().load("https://lh3.googleu
const cloudMaterial1 = new THREE.SpriteMaterial({
  map: cloudTexture1,
  transparent: true,
  opacity: 0.05
});
const cloud1 = new THREE.Sprite(cloudMaterial1);
cloud1.position.set(Math.random() * 10 - 7, 3 + Math.random() * 2, 0);
cloud1.scale.set(24, 8, 1);
scene.add(cloud1);

// Add Clouds (Second Cloud)
const cloudTexture2 = new THREE.TextureLoader().load("https://static.vect
const cloudMaterial2 = new THREE.SpriteMaterial({
  map: cloudTexture2,
  transparent: true,
  opacity: 0.2
});
```

Successes, Failures, and What Can Be Improved

Successes

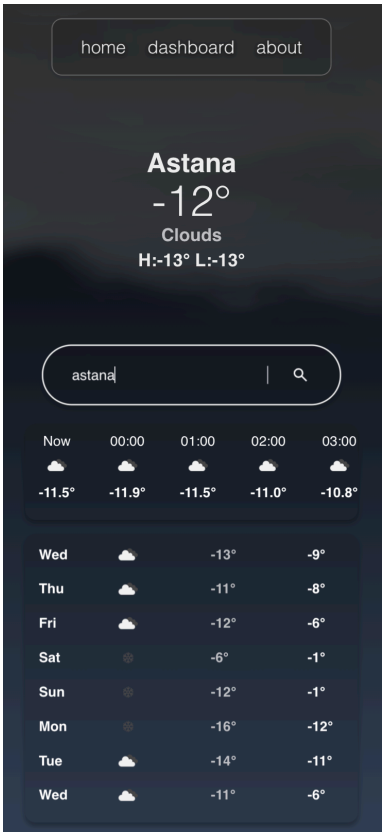
Dashboard updates: during the first presentation, I didn't finish creating functionality for the main dashboard. I wanted to add search, overlay on clicking, and advanced animations. Below is comparison between the first and the final design and functionality for the dashboard.

before



after

Additionally, I created mobile adaptability for the website:



Failures

I was unable to create a feature showcasing the top cities by air quality due to the lack of accessible APIs. While researching, I found that most reliable air quality APIs require an enterprise-level subscription to access the necessary data. Unfortunately, this was beyond the scope and resources of the project.

Adding second and third grid functionalities proved to be challenging. While I initially planned to include this feature, handling the complexity of the structure and ensuring a responsive design became increasingly difficult within the limited timeframe of one week. As a result, I decided to focus on maintaining the overall stability and usability of the project instead of rushing to implement a poorly structured feature.

What Can Be Improved

Explore Alternative APIs: Research more affordable or free APIs to provide additional features, like:

• Global air quality comparisons.

• Historical air quality data.

• Implement Fallbacks: If primary data sources fail, consider adding alternative APIs as backups to improve reliability.

Localization Support: Allow users to switch between languages for global accessibility.

• User Preferences: Add features to let users customize the dashboard, such as selecting cities or data types to display.

Cloud Integration: Use services like AWS, Firebase, or Heroku to host the project and enable future scalability.

Conclusion

In conclusion, this project aimed to provide an engaging and informative user experience by displaying real-time data on air quality, weather, and relevant environmental factors. Despite encountering some challenges, such as limitations with API access for top cities by air quality and the complexity of implementing a second and third grid layout within the given timeframe, the project successfully showcases the core functionality with a responsive and interactive design.

The integration of key libraries and frameworks like Three.js for the rain animation, Tailwind for styling, and various APIs such as Google Air Quality, OpenWeather, and Windy has allowed for a smooth and dynamic interface. However, there are opportunities to further enhance the project with additional features, such as improved data visualization, dynamic grids, and real-time updates, which can be explored in future iterations.

While some features were scaled back or adjusted to ensure a stable and functional project, the overall foundation is solid, providing a great starting point for continued development. The feedback and lessons learned from this project will guide future improvements, ensuring the app can grow in both functionality and performance.

This project demonstrates the potential of combining real-time environmental data with an interactive interface, and with further enhancements, it could become a powerful tool for users seeking to monitor air quality and weather conditions in their regions.