ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО	*УТВЕРЖДАЮ*
Доцент департамента программной инженерии	Академический руководитель образовательной программы «Программная инженерия»
_____ Р.А. Родригес Залепинос	_____ Н.А. Павлочев
«_____» _____ 2024 г.	«_____» _____ 2024 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Программа для визуализации данных о климате и погоде помощью JavaScript.

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.06-01 81 01-1-ЛУ

Исполнитель: студент 4 курса
программы «Программная инженерия»
_____ Д.А. Щербаков
«_____» _____ 2024 г.

Москва 2024

Содержание

1 Текст программы

1.1 src/index.tsx

```
import { createRoot } from "react-dom/client";
import { App, ConfigProvider } from "antd";

import { Layout } from "~/components/Layout";

import "antd/dist/reset.css";

import "~/worker";
import "~/atoms/dataset";
import "~/index.css";

/**
 * Root component
 */
export function Root() {
  return (
    <ConfigProvider>
      <App>
        <Layout />
      </App>
    </ConfigProvider>
  );
}

createRoot(document.getElementById("root")!).render(<Root />);
```

1.2 src/worker/worker.ts

```
import * as idb from "idb-keyval";
import { Request, Response, Error } from "~/worker";
import { loadGoesData } from "~/loaders/goes";
import { h3bin } from "~/utils/h3bin";
import { loadEra5Data } from "~/loaders/era5";
import { gridBin } from "~/utils/grid";
import { contour } from "~/utils/contour";
```

```
/**
 * Size of each element in the buffer
 */
export const elementSize = {
  h3: 2 * Int32Array.BYTES_PER_ELEMENT + Float32Array.BYTES_PER_ELEMENT,
  grid: 3 * Float32Array.BYTES_PER_ELEMENT,
};

/**
 * A callback function to post a response to the main thread and store th
 */
function onLoad(response: Response | Error) {
  postMessage(response);

  if ("error" in response) {
    return;
  }

  if (response.type === "h3" || response.type === "grid") {
    const length = response.count * elementSize[response.type];

    const destination = new Uint8ClampedArray(length);
    const source = new Uint8ClampedArray(response.buffer, 0, length);
    destination.set(source);

    idb.set(response.key, {
      ...response,
      buffer: destination.buffer,
    });
  } else {
    idb.set(response.key, { ...response, buffer: undefined });
  }
}

addEventListener("message", async ({ data: request }: { data: Request })
  if (!self.document) {
    // @ts-ignore
    self.document = { currentScript: {} }; // hack for worker
```

```
}

const { path, buffer, source, type, variable } = request;

try {
  let data, result;
  switch (source) {
    case "ERA5":
      data = await loadEra5Data(path, variable);
      break;
    case "GOES-16":
      data = await loadGoesData(path, variable);
      break;
    default:
      data = null;
  }

  if (!data) {
    return onLoad({ ...request, error: "Data error" });
  }

  const { min, max } = data;

  switch (type) {
    case "h3":
      result = h3bin(data.data, buffer, { resolution: 4 });
      break;
    case "grid":
      result = gridBin(data.data, buffer, { resolution: 1000 });
      break;
    case "contour":
      result = contour(data.data, { min, max, breaks: 8 });
      break;
    default:
      result = null;
  }

  if (result) {
    onLoad({
```

```
        ...result,
        ...request,
        min,
        max,
        date: new Date(),
        variables: data.variables,
      } as Response);
    }
  } catch (error) {
    onLoad({ ...request, error });
  }
});
```

1.3 src/worker/index.ts

```
import type { DatasetResult, DatasetParams } from "~/atoms/dataset";

/**
 * Request to load a dataset
 */
export type Request = DatasetParams & {
  buffer: SharedArrayBuffer | ArrayBuffer;
};

/**
 * Response from the worker in case of error
 */
export type Error = DatasetParams & { error: any };

/**
 * Response from the worker
 */
export type Response = DatasetParams &
  DatasetResult & {
    buffer: SharedArrayBuffer | ArrayBuffer;
  };

/**
 * Worker instance
```

```
 */
const worker = new Worker(new URL("./worker.ts", import.meta.url), {
  type: "module",
});

const resolvers: Record<string, (response: Response) => void> = {};
const rejectors: Record<string, (response: Error) => void> = {};

worker.addEventListener("message", ({ data }: { data: Response | Error }) =>
  if ("error" in data) {
    rejectors[data.key] && rejectors[data.key](data);
  } else {
    resolvers[data.key] && resolvers[data.key](data);
  }
});

/**
 * Load a dataset using a worker
 */
export function load(request: Request): Promise<Response> {
  worker.postMessage(request);

  return new Promise((resolve, reject) => {
    resolvers[request.key] = resolve;
    rejectors[request.key] = reject;
  });
}
```

1.4 src/utils/layer.ts

```
import * as h3 from "h3-js";
import chroma from "chroma-js";
import { Dayjs } from "dayjs";
import { GridLayer, ContourLayer } from "@deck.gl/aggregation-layers/type
import { H3HexagonLayer } from "@deck.gl/geo-layers/typed";
import { GeoJsonLayer } from "@deck.gl/layers/typed";
import { Layer as DeckGLLayer } from "@deck.gl/core/typed";

import { LayerSettings } from "~/atoms/layer";
```

```
import { Dataset, DatasetParams } from "~/atoms/dataset";
import { colors } from "~/utils/colors";

/**
 * calculate parameters to pass to the loader
 */
export function getParams(
  layer: Partial<LayerSettings>,
  date: Dayjs
): DatasetParams | undefined {
  if (!layer || !layer.product || !layer.type) {
    return;
  }
  const { product, type } = layer;
  const [source, variable] = product.split("/");
  const key =
    `${type}/${product}/${variable}` +
    `/${date.format("YYYY/MM/DD/HH:mm")}` +
    (layer.variable ? `/${layer.variable}` : "");

  if (source.startsWith("GOES-16")) {
    const year = `${date.year()}`;
    const day = `${date.diff(date.startOf("year"), "day")}`.padStart(3, "
    const hour = `${date.hour()}`.padStart(2, "0");
    const path = `${variable}/${year}/${day}/${hour}`;

    return {
      type,
      key,
      path,
      variable: layer.variable,
      source: "GOES-16" as const,
    };
  } else if (source.includes("era5")) {
    const year = `${date.year()}`;
    const month = `${date.month()}`.padStart(2, "0");
    const day = `${date.date()}`.padStart(2, "0");
    const time = date.format("HH:mm");
    const path = `${source}/${variable}/${year}/${month}/${day}/${time}`;
```

```
    return {
      type,
      key,
      path,
      variable: layer.variable,
      source: "ERA5" as const,
    };
  }
}

const layers = new Map<string, DeckGLLayer>();

/**
 * instantiate a deck.gl layer to render a dataset
 */
export function getDeckGlLayer(
  dataset: Dataset,
  layerSettings: LayerSettings
): DeckGLLayer | undefined {
  const { palette, opacity, visible } = layerSettings;
  const key = `${dataset.key}/${layerSettings.key}/${palette}`;
  const path = palette.split(".");
  const colorScale = colors[path[0]][path[1]];
  const { min, max } = dataset;
  const scale = chroma.scale(colorScale).domain([min, max]);

  if (!layers.has(key)) {
    if (dataset.type === "h3") {
      const { buffer, count } = dataset;
      const offset = Int32Array.BYTES_PER_ELEMENT;
      const step = offset * 2 + Float32Array.BYTES_PER_ELEMENT;
      const length = count * step;
      const view = new DataView(buffer, 0, length);

      function* getData() {
        for (let i = 0; i < count; ++i) {
          const left = view.getInt32(step * i);
          const right = view.getInt32(step * i + offset);
```

```
        const value = view.getFloat32(step * i + offset * 2);
        yield { hexagon: h3.splitLongToH3Index(left, right), value };
      }
    }

    const data = [...getData()];

    const deckGlLayer = new H3HexagonLayer({
      id: key,
      filled: true,
      pickable: true,
      blend: true,
      opacity: opacity,
      visible: visible,
      data: data,
      wireframe: false,
      getHexagon: (item: { hexagon: string }) => {
        return item.hexagon;
      },
      getFillColor: (item: { value: number }) => {
        return scale(item.value).rgb();
      },
    });

    layers.set(key, deckGlLayer);
  } else if (dataset.type === "grid") {
    const { buffer, count } = dataset;
    const view = new Float32Array(buffer, 0, count * 3);
    const colorRange = scale.colors(16, null).map((color) => color.rgb(

    const deckGlLayer = new GridLayer({
      id: key,
      cellSize: 40000,
      pickable: true,
      extruded: false,
      opacity: opacity,
      visible: visible,
      colorRange: colorRange as any,
      colorDomain: [min, max],
```

```
        colorAggregation: "MEAN",
        data: { length: count },
        gpuAggregation: true,
        getPosition: (_, { index }) => {
          return [view[index * 3], view[index * 3 + 1]];
        },
        getColorWeight: (_, { index }) => {
          return view[index * 3 + 2];
        },
      });

      layers.set(key, deckGlLayer);
    } else if (dataset.type === "contour") {
      const deckGlLayer = new GeoJsonLayer({
        id: key,
        data: dataset.features,
        getLineWidth: 2,
        getLineColor: (line: any) => {
          const value: number = line.properties.value;
          return scale(value).rgb();
        },
        lineWidthUnits: "pixels",
        filled: true,
        stroked: true,
        pickable: false,
        wrapLongitude: true,
      });

      layers.set(key, deckGlLayer);
    }
  } else {
    const deckGlLayer = layers.get(key)!.clone({ visible, opacity });
    layers.set(key, deckGlLayer);
  }

  return layers.get(key);
}
```

1.5 src/utils/h3bin.ts

```ts
import * as h3 from "h3-js";

/**
 * create a hexagonal grid from a list of points
 */
export function h3bin(
  data: [number, number, number][],
  buffer: SharedArrayBuffer | ArrayBuffer,
  { resolution }: { resolution: number }
) {
  const cells = new Map<string, [number, number]>();

  for (const [lon, lat, value] of data) {
    if (!value) {
      continue;
    }
    const index = h3.latLngToCell(lat, lon, resolution);
    const [sum, count] = cells.get(index) || [0, 0];
    cells.set(index, [sum + value, count + 1]);
  }

  const view = new DataView(buffer);

  const offset = Int32Array.BYTES_PER_ELEMENT;
  const step = offset * 2 + Float32Array.BYTES_PER_ELEMENT;

  let index = 0;
  for (const [cell, [sum, count]] of cells) {
    const [left, right] = h3.h3IndexToSplitLong(cell);
    view.setInt32(index * step, left);
    view.setInt32(index * step + offset, right);
    view.setFloat32(index * step + offset * 2, sum / count);
    ++index;
  }

  return { type: "h3" as const, resolution, buffer, count: index };
}
```

1.6 src/utils/grid.ts

```ts
/**
 * convert array of [lon, lat, value] to binary grid
 */
export function gridBin(
  data: [number, number, number][],
  buffer: SharedArrayBuffer | ArrayBuffer,
  { resolution }: { resolution: number }
) {
  const view = new Float32Array(buffer, 0, data.length * 3);

  for (let i = 0; i < data.length; ++i) {
    const [lon, lat, value] = data[i];
    if (!value) {
      continue;
    }
    view[i * 3] = lon;
    view[i * 3 + 1] = lat;
    view[i * 3 + 2] = value;
  }

  return {
    buffer,
    resolution,
    count: data.length,
    type: "grid" as const,
  };
}
```

1.7 src/utils/contour.ts

```ts
import { geojsonToBinary } from "@loaders.gl/gis";
import * as turf from "@turf/turf";

/**
 * Create isolines from a grid of points.
 */
export function contour(
```

```
  data: [number, number, number][],
  { min, max, breaks }: { min: number; max: number; breaks: number }
) {
  const points = [];
  for (const point of data) {
    points.push(turf.point([point[0], point[1]], { value: point[2] }));
  }

  const bands = Array.from(
    { length: breaks },
    (_, i) => min + (max - min) * (i / 10)
  );

  const fc = turf.featureCollection(points);

  const isolines = turf.isolines(fc, bands, {
    zProperty: "value",
  });

  const features = geojsonToBinary(isolines.features, {
    PositionDataType: Float32Array,
    fixRingWinding: true,
  });

  return { features };
}
```

1.8 src/utils/colors.ts

```
import chroma from "chroma-js";

/**
 * Color palettes for use in charts.
 */
export const colors = {
  Sequential: {
    Viridis: [
      "#440154",
      "#482878",
```

```
    "#3e4989",
    "#31688e",
    "#26828e",
    "#1f9e89",
    "#35b779",
    "#6ece58",
    "#b5de2b",
    "#fde725",
  ],
  Cividis: [
    "#00224e",
    "#123570",
    "#3b496c",
    "#575d6d",
    "#707173",
    "#8a8678",
    "#a59c74",
    "#c3b369",
    "#e1cc55",
    "#fee838",
  ],
  Inferno: [
    "#000004",
    "#1b0c41",
    "#4a0c6b",
    "#781c6d",
    "#a52c60",
    "#cf4446",
    "#ed6925",
    "#fb9b06",
    "#f7d13d",
    "#fcffa4",
  ],
  Magma: [
    "#000004",
    "#180f3d",
    "#440f76",
    "#721f81",
    "#9e2f7f",
```

```
      "#cd4071",
      "#f1605d",
      "#fd9668",
      "#feca8d",
      "#fcfdbf",
    ],
    Plasma: [
      "#0d0887",
      "#46039f",
      "#7201a8",
      "#9c179e",
      "#bd3786",
      "#d8576b",
      "#ed7953",
      "#fb9f3a",
      "#fdca26",
      "#f0f921",
    ],
    Purples: chroma.brewer["Purples"],
    Blues: chroma.brewer["Blues"],
    Greens: chroma.brewer["Greens"],
    Oranges: chroma.brewer["Oranges"],
    Reds: chroma.brewer["Reds"],
    YlOrBr: chroma.brewer["YlOrBr"],
    YlOrRd: chroma.brewer["YlOrRd"],
    OrRd: chroma.brewer["OrRd"],
    PuRd: chroma.brewer["PuRd"],
    RdPu: chroma.brewer["RdPu"],
    BuPu: chroma.brewer["BuPu"],
    PuBu: chroma.brewer["PuBu"],
    PuBuGn: chroma.brewer["PuBuGn"],
    GnBu: chroma.brewer["GnBu"],
    BuGn: chroma.brewer["BuGn"],
    YlGnBu: chroma.brewer["YlGnBu"],
    YlGn: chroma.brewer["YlGn"],
    Greys: chroma.brewer["Greys"],
  },
  Divergent: {
    Spectral: chroma.brewer["Spectral"].reverse(),
```

```
    RdYlGn: chroma.brewer["RdYlGn"].reverse(),
    RdBu: chroma.brewer["RdBu"].reverse(),
    PiYG: chroma.brewer["PiYG"].reverse(),
    PRGn: chroma.brewer["PRGn"].reverse(),
    RdYlBu: chroma.brewer["RdYlBu"].reverse(),
    BrBG: chroma.brewer["BrBG"].reverse(),
    RdGy: chroma.brewer["RdGy"].reverse(),
  },
  Categorical: {
    D3: [
      "#1f77b4",
      "#ff7f0e",
      "#2ca02c",
      "#d62728",
      "#9467bd",
      "#8c564b",
      "#e377c2",
      "#7f7f7f",
      "#bcbd22",
      "#17becf",
    ],
    G10: [
      "#3366cc",
      "#dc3912",
      "#ff9900",
      "#109618",
      "#990099",
      "#0099c6",
      "#dd4477",
      "#66aa00",
      "#b82e2e",
      "#316395",
    ],
    T10: [
      "#4c78a8",
      "#f58518",
      "#e45756",
      "#72b7b2",
      "#54a24b",
```

```
    "#eeca3b",
    "#b279a2",
    "#ff9da6",
    "#9d755d",
    "#bab0ac",
  ],
  Alphabet: [
    "#AA0DFE",
    "#3283FE",
    "#85660D",
    "#782AB6",
    "#565656",
    "#1C8356",
    "#16FF32",
    "#F7E1A0",
    "#E2E2E2",
    "#1CBE4F",
    "#C4451C",
    "#DEA0FD",
    "#FE00FA",
    "#325A9B",
    "#FEAF16",
    "#F8A19F",
    "#90AD1C",
    "#F6222E",
    "#1CFFCE",
    "#2ED9FF",
    "#B10DA1",
    "#C075A6",
    "#FC1CBF",
    "#B00068",
    "#FBE426",
    "#FA0087",
  ],
  Dark24: [
    "#2E91E5",
    "#E15F99",
    "#1CA71C",
    "#FB0D0D",
```

```
    "#DA16FF",
    "#222A2A",
    "#B68100",
    "#750D86",
    "#EB663B",
    "#511CFB",
    "#00A08B",
    "#FB00D1",
    "#FC0080",
    "#B2828D",
    "#6C7C32",
    "#778AAE",
    "#862A16",
    "#A777F1",
    "#620042",
    "#1616A7",
    "#DA60CA",
    "#6C4516",
    "#0D2A63",
    "#AF0038",
  ],
  Light24: [
    "#FD3216",
    "#00FE35",
    "#6A76FC",
    "#FED4C4",
    "#FE00CE",
    "#0DF9FF",
    "#F6F926",
    "#FF9616",
    "#479B55",
    "#EEA6FB",
    "#DC587D",
    "#D626FF",
    "#6E899C",
    "#00B5F7",
    "#B68E00",
    "#C9FBE5",
    "#FF0092",
```

```
      "#22FFA7",
      "#E3EE9E",
      "#86CE00",
      "#BC7196",
      "#7E7DCD",
      "#FC6955",
      "#E48F72",
    ],
    Set1: chroma.brewer["Set1"],
    Pastel1: chroma.brewer["Pastel1"],
    Dark2: chroma.brewer["Dark2"],
    Set2: chroma.brewer["Set2"],
    Pastel2: chroma.brewer["Pastel2"],
    Set3: chroma.brewer["Set3"],
  },
  Cyclical: {
    Twilight: [
      "#e2d9e2",
      "#9ebbc9",
      "#6785be",
      "#5e43a5",
      "#421257",
      "#471340",
      "#8e2c50",
      "#ba6657",
      "#ceac94",
      "#e2d9e2",
    ],
    IceFire: [
      "#000000",
      "#001f4d",
      "#003786",
      "#0e58a8",
      "#217eb8",
      "#30a4ca",
      "#54c8df",
      "#9be4ef",
      "#e1e9d1",
      "#f3d573",
```

```
    "#e7b000",
    "#da8200",
    "#c65400",
    "#ac2301",
    "#820000",
    "#4c0000",
    "#040100",
  ],
  Edge: [
    "#313131",
    "#3d019d",
    "#3810dc",
    "#2d47f9",
    "#2593ff",
    "#2adef6",
    "#60fdfa",
    "#aefdff",
    "#f3f3f1",
    "#fffda9",
    "#fafd5b",
    "#f7da29",
    "#ff8e25",
    "#f8432d",
    "#d90d39",
    "#97023d",
    "#313131",
  ],
  Phase: [
    "rgb(167, 119, 12)",
    "rgb(197, 96, 51)",
    "rgb(217, 67, 96)",
    "rgb(221, 38, 163)",
    "rgb(196, 59, 224)",
    "rgb(153, 97, 244)",
    "rgb(95, 127, 228)",
    "rgb(40, 144, 183)",
    "rgb(15, 151, 136)",
    "rgb(39, 153, 79)",
    "rgb(119, 141, 17)",
```

```
    "rgb(167, 119, 12)",
  ],
  HSV: [
    "#ff0000",
    "#ffa700",
    "#afff00",
    "#08ff00",
    "#00ff9f",
    "#00b7ff",
    "#0010ff",
    "#9700ff",
    "#ff00bf",
    "#ff0018",
  ],
  mrybm: [
    "#f884f7",
    "#f968c4",
    "#ea4388",
    "#cf244b",
    "#b51a15",
    "#bd4304",
    "#cc6904",
    "#d58f04",
    "#cfaa27",
    "#a19f62",
    "#588a93",
    "#2269c4",
    "#3e3ef0",
    "#6b4ef9",
    "#956bfa",
    "#cd7dfe",
  ],
  mygbm: [
    "#ef55f1",
    "#fb84ce",
    "#fbafa1",
    "#fcd471",
    "#f0ed35",
    "#c6e516",
```

```
    "#96d310",
    "#61c10b",
    "#31ac28",
    "#439064",
    "#3d719a",
    "#284ec8",
    "#2e21ea",
    "#6324f5",
    "#9139fa",
    "#c543fa",
  ],
},
Constant: {
  White: ["#ffffff"],
  Black: ["#000000"],
},
Oceanography: {
  turbid: [
    "rgb(232, 245, 171)",
    "rgb(220, 219, 137)",
    "rgb(209, 193, 107)",
    "rgb(199, 168, 83)",
    "rgb(186, 143, 66)",
    "rgb(170, 121, 60)",
    "rgb(151, 103, 58)",
    "rgb(129, 87, 56)",
    "rgb(104, 72, 53)",
    "rgb(80, 59, 46)",
    "rgb(57, 45, 37)",
    "rgb(34, 30, 27)",
  ],
  thermal: [
    "rgb(3, 35, 51)",
    "rgb(13, 48, 100)",
    "rgb(53, 50, 155)",
    "rgb(93, 62, 153)",
    "rgb(126, 77, 143)",
    "rgb(158, 89, 135)",
    "rgb(193, 100, 121)",
```

```
    "rgb(225, 113, 97)",
    "rgb(246, 139, 69)",
    "rgb(251, 173, 60)",
    "rgb(246, 211, 70)",
    "rgb(231, 250, 90)",
  ],
  haline: [
    "rgb(41, 24, 107)",
    "rgb(42, 35, 160)",
    "rgb(15, 71, 153)",
    "rgb(18, 95, 142)",
    "rgb(38, 116, 137)",
    "rgb(53, 136, 136)",
    "rgb(65, 157, 133)",
    "rgb(81, 178, 124)",
    "rgb(111, 198, 107)",
    "rgb(160, 214, 91)",
    "rgb(212, 225, 112)",
    "rgb(253, 238, 153)",
  ],
  solar: [
    "rgb(51, 19, 23)",
    "rgb(79, 28, 33)",
    "rgb(108, 36, 36)",
    "rgb(135, 47, 32)",
    "rgb(157, 66, 25)",
    "rgb(174, 88, 20)",
    "rgb(188, 111, 19)",
    "rgb(199, 137, 22)",
    "rgb(209, 164, 32)",
    "rgb(217, 192, 44)",
    "rgb(222, 222, 59)",
    "rgb(224, 253, 74)",
  ],
  ice: [
    "rgb(3, 5, 18)",
    "rgb(25, 25, 51)",
    "rgb(44, 42, 87)",
    "rgb(58, 60, 125)",
```

```
    "rgb(62, 83, 160)",
    "rgb(62, 109, 178)",
    "rgb(72, 134, 187)",
    "rgb(89, 159, 196)",
    "rgb(114, 184, 205)",
    "rgb(149, 207, 216)",
    "rgb(192, 229, 232)",
    "rgb(234, 252, 253)",
  ],
  gray: [
    "rgb(0, 0, 0)",
    "rgb(16, 16, 16)",
    "rgb(38, 38, 38)",
    "rgb(59, 59, 59)",
    "rgb(81, 80, 80)",
    "rgb(102, 101, 101)",
    "rgb(124, 123, 122)",
    "rgb(146, 146, 145)",
    "rgb(171, 171, 170)",
    "rgb(197, 197, 195)",
    "rgb(224, 224, 223)",
    "rgb(254, 254, 253)",
  ],
  oxy: [
    "rgb(63, 5, 5)",
    "rgb(101, 6, 13)",
    "rgb(138, 17, 9)",
    "rgb(96, 95, 95)",
    "rgb(119, 118, 118)",
    "rgb(142, 141, 141)",
    "rgb(166, 166, 165)",
    "rgb(193, 192, 191)",
    "rgb(222, 222, 220)",
    "rgb(239, 248, 90)",
    "rgb(230, 210, 41)",
    "rgb(220, 174, 25)",
  ],
  deep: [
    "rgb(253, 253, 204)",
```

```
    "rgb(206, 236, 179)",
    "rgb(156, 219, 165)",
    "rgb(111, 201, 163)",
    "rgb(86, 177, 163)",
    "rgb(76, 153, 160)",
    "rgb(68, 130, 155)",
    "rgb(62, 108, 150)",
    "rgb(62, 82, 143)",
    "rgb(64, 60, 115)",
    "rgb(54, 43, 77)",
    "rgb(39, 26, 44)",
  ],
  dense: [
    "rgb(230, 240, 240)",
    "rgb(191, 221, 229)",
    "rgb(156, 201, 226)",
    "rgb(129, 180, 227)",
    "rgb(115, 154, 228)",
    "rgb(117, 127, 221)",
    "rgb(120, 100, 202)",
    "rgb(119, 74, 175)",
    "rgb(113, 50, 141)",
    "rgb(100, 31, 104)",
    "rgb(80, 20, 66)",
    "rgb(54, 14, 36)",
  ],
  algae: [
    "rgb(214, 249, 207)",
    "rgb(186, 228, 174)",
    "rgb(156, 209, 143)",
    "rgb(124, 191, 115)",
    "rgb(85, 174, 91)",
    "rgb(37, 157, 81)",
    "rgb(7, 138, 78)",
    "rgb(13, 117, 71)",
    "rgb(23, 95, 61)",
    "rgb(25, 75, 49)",
    "rgb(23, 55, 35)",
    "rgb(17, 36, 20)",
```

```
],
matter: [
  "rgb(253, 237, 176)",
  "rgb(250, 205, 145)",
  "rgb(246, 173, 119)",
  "rgb(240, 142, 98)",
  "rgb(231, 109, 84)",
  "rgb(216, 80, 83)",
  "rgb(195, 56, 90)",
  "rgb(168, 40, 96)",
  "rgb(138, 29, 99)",
  "rgb(107, 24, 93)",
  "rgb(76, 21, 80)",
  "rgb(47, 15, 61)",
],
speed: [
  "rgb(254, 252, 205)",
  "rgb(239, 225, 156)",
  "rgb(221, 201, 106)",
  "rgb(194, 182, 59)",
  "rgb(157, 167, 21)",
  "rgb(116, 153, 5)",
  "rgb(75, 138, 20)",
  "rgb(35, 121, 36)",
  "rgb(11, 100, 44)",
  "rgb(18, 78, 43)",
  "rgb(25, 56, 34)",
  "rgb(23, 35, 18)",
],
amp: [
  "rgb(241, 236, 236)",
  "rgb(230, 209, 203)",
  "rgb(221, 182, 170)",
  "rgb(213, 156, 137)",
  "rgb(205, 129, 103)",
  "rgb(196, 102, 73)",
  "rgb(186, 74, 47)",
  "rgb(172, 44, 36)",
  "rgb(149, 19, 39)",
```

```
    "rgb(120, 14, 40)",
    "rgb(89, 13, 31)",
    "rgb(60, 9, 17)",
  ],
  tempo: [
    "rgb(254, 245, 244)",
    "rgb(222, 224, 210)",
    "rgb(189, 206, 181)",
    "rgb(153, 189, 156)",
    "rgb(110, 173, 138)",
    "rgb(65, 157, 129)",
    "rgb(25, 137, 125)",
    "rgb(18, 116, 117)",
    "rgb(25, 94, 106)",
    "rgb(28, 72, 93)",
    "rgb(25, 51, 80)",
    "rgb(20, 29, 67)",
  ],
  phase: [
    "rgb(167, 119, 12)",
    "rgb(197, 96, 51)",
    "rgb(217, 67, 96)",
    "rgb(221, 38, 163)",
    "rgb(196, 59, 224)",
    "rgb(153, 97, 244)",
    "rgb(95, 127, 228)",
    "rgb(40, 144, 183)",
    "rgb(15, 151, 136)",
    "rgb(39, 153, 79)",
    "rgb(119, 141, 17)",
    "rgb(167, 119, 12)",
  ],
  balance: [
    "rgb(23, 28, 66)",
    "rgb(41, 58, 143)",
    "rgb(11, 102, 189)",
    "rgb(69, 144, 185)",
    "rgb(142, 181, 194)",
    "rgb(210, 216, 219)",
```

```
      "rgb(230, 210, 204)",
      "rgb(213, 157, 137)",
      "rgb(196, 101, 72)",
      "rgb(172, 43, 36)",
      "rgb(120, 14, 40)",
      "rgb(60, 9, 17)",
    ],
    delta: [
      "rgb(16, 31, 63)",
      "rgb(38, 62, 144)",
      "rgb(30, 110, 161)",
      "rgb(60, 154, 171)",
      "rgb(140, 193, 186)",
      "rgb(217, 229, 218)",
      "rgb(239, 226, 156)",
      "rgb(195, 182, 59)",
      "rgb(115, 152, 5)",
      "rgb(34, 120, 36)",
      "rgb(18, 78, 43)",
      "rgb(23, 35, 18)",
    ],
    curl: [
      "rgb(20, 29, 67)",
      "rgb(28, 72, 93)",
      "rgb(18, 115, 117)",
      "rgb(63, 156, 129)",
      "rgb(153, 189, 156)",
      "rgb(223, 225, 211)",
      "rgb(241, 218, 206)",
      "rgb(224, 160, 137)",
      "rgb(203, 101, 99)",
      "rgb(164, 54, 96)",
      "rgb(111, 23, 91)",
      "rgb(51, 13, 53)",
    ],
  },
} as Record<string, Record<string, string[]>>;
```

1.9 src/loaders/utils.ts

```
import proj4 from "proj4";
import type { Dataset, File } from "h5wasm";

/**
 * Extract value from HDF5/netCDF4 dataset.
 */
export function getValue(file: File, path: string) {
  const dataset = file.get(path) as Dataset;
  const [offset] = dataset.attrs["add_offset"]
    ? (dataset.attrs["add_offset"].value as Float32Array)
    : [0];
  const [scale] = dataset.attrs["scale_factor"]
    ? (dataset.attrs["scale_factor"].value as Float32Array)
    : [1];
  const value = dataset.value as Float32Array;

  if (!dataset.attrs["_FillValue"]) {
    return { offset, scale, value };
  }

  const [fill] = dataset.attrs["_FillValue"].value as Float32Array;

  return { offset, scale, fill, value };
}

/**
 * Extract projection from geostaionary satellite data.
 */
export function getGeosProjection(file: File) {
  const goesImagerProjection = file.get("goes_imager_projection") as Data
  const [longitudeOfProjectionOrigin] = goesImagerProjection.attrs[
    "longitude_of_projection_origin"
  ].value as Float32Array;
  const [perspectivePointHeight] = goesImagerProjection.attrs[
    "perspective_point_height"
  ].value as Float32Array;
  const [semiMinorAxis] = goesImagerProjection.attrs["semi_minor_axis"]
    .value as Float32Array;
```

```
  const [semiMajorAxis] = goesImagerProjection.attrs["semi_major_axis"]
    .value as Float32Array;
  const sweepAngleAxis = goesImagerProjection.attrs["sweep_angle_axis"]
    .value as string;

  const projection = proj4(
    `+proj=geos +sweep=${sweepAngleAxis} +lon_0=${longitudeOfProjectionOr
  );

  return {
    projection,
    perspectivePointHeight,
    semiMajorAxis,
    semiMinorAxis,
    longitudeOfProjectionOrigin,
    sweepAngleAxis,
  };
}
```

1.10 src/loaders/modis.ts

1.11 src/loaders/goes.ts

```
import h5wasm, { Dataset } from "h5wasm";

import { S3Client, ListObjectsV2Command } from "@aws-sdk/client-s3";
import { getValue, getGeosProjection } from "~/loaders/utils";

const client = new S3Client({
  region: "us-east-1",
  signer: { sign: async (request) => request },
});

/**
 * Load GOES data from S3 bucket.
 */
export async function loadGoesData(path: string, initialVariable?: string
```

```
const { FS } = await h5wasm.ready;

const list = await client.send(
  new ListObjectsV2Command({
    Bucket: "noaa-goes16",
    Prefix: path,
  })
);

const filePath = list.Contents![0].Key!;
const name = path.split("/").at(-1) as string;

const data = await new Promise<ArrayBuffer>((resolve, reject) => {
  const makeFetch = (tries = 5) =>
    fetch(`https://noaa-goes16.s3.amazonaws.com/${filePath}`)
      .then((data) => data.arrayBuffer())
      .then((data) => resolve(data))
      .catch((error) => {
        if (tries > 0) {
          setTimeout(() => makeFetch(tries - 1), 5000);
        } else {
          reject(error);
        }
      });
  makeFetch();
});

FS.writeFile(name, new Uint8Array(data));

const file = new h5wasm.File(name, "r");
const result: [number, number, number][] = [];

const variables: string[] = [];
for (const [name, variable] of file.items()) {
  if (
    !variable ||
    !(typeof variable === "object") ||
    !("attrs" in variable)
  ) {
```

```
      continue;
    }
    if (
      variable.attrs["grid_mapping"] &&
      variable.attrs["grid_mapping"].value === "goes_imager_projection" &
      "shape" in variable &&
      variable.shape.length === 2
    ) {
      variables.push(name as string);
    }
}

const variable =
  initialVariable && variables.includes(initialVariable)
    ? initialVariable
    : variables[0];
const { value: X, offset: xOffset, scale: xScale } = getValue(file, "x"
const { value: Y, offset: yOffset, scale: yScale } = getValue(file, "y"
const { value: values, offset, scale, fill } = getValue(file, variable)
const { projection, perspectivePointHeight } = getGeosProjection(file);
let min = Infinity;
let max = -Infinity;

for (let i = 0; i < X.length; ++i) {
  for (let j = 0; j < Y.length; ++j) {
    const index = X.length * j + i;

    if (values[index] === fill) {
      continue;
    }

    const value = values[index] * scale + offset;

    if (Number.isNaN(value)) {
      continue;
    }
    if (!(values[index] === fill || Number.isNaN(value))) {
      if (value < min) {
        min = value;
```

```
        }
        if (value > max) {
          max = value;
        }
      }

      const x = (X[i] * xScale + xOffset) * perspectivePointHeight;
      const y = (Y[j] * yScale + yOffset) * perspectivePointHeight;
      const coords = projection.inverse([x, y]) as [number, number];
      if (Number.isNaN(coords[0]) || Number.isNaN(coords[1])) {
        continue;
      }

      result.push([...coords, value]);
    }
  }

  file.close();

  return { data: result, variables, min, max };
}
```

1.12 src/loaders/era5.ts

```
import { unzipSync } from "fflate";
import { NetCDFReader } from "@loaders.gl/netcdf";

/**
 * Load ERA5 data from the CDS API using proxy worker.
 */
export async function loadEra5Data(
  path: string,
  initialVariable?: string
): Promise<{
  data: [number, number, number][];
  variables: string[];
  min: number;
  max: number;
}> {
```

```
const buffer = await fetch(`/api/cds/${path}`).then(async (response) =>
  if (response.ok) {
    return await response.arrayBuffer();
  } else {
    throw await response.json();
  }
});
const file = unzipSync(new Uint8Array(buffer))["data.nc"];
const data = new NetCDFReader(file);

const variables = data.variables.filter(
  (variable) => !["longitude", "latitude", "time"].includes(variable.na
);
const variable =
  variables.find((variable) => variable.name === initialVariable) ||
  variables[0];
const variableNames = variables.map((variable) => variable.name);

if (!variable) {
  throw {};
}

const result: [number, number, number][] = [];
let min = 50000;
let max = -50000;

const values = data.getDataVariable(variable.name);
const longitude = data.getDataVariable("longitude");
const latitude = data.getDataVariable("latitude");

const { fill, offset, scale } = (
  variable.attributes as Record<string, number | string>[]
).reduce((acc, curr) => {
  if (curr.name === "_FillValue") {
    acc["fill"] = curr.value as number;
  } else if (curr.name === "add_offset") {
    acc["offset"] = curr.value as number;
  } else if (curr.name === "scale_factor") {
    acc["scale"] = curr.value as number;
```

```
    }
    return acc;
  }, {}) as { fill: number; offset: number; scale: number };

  for (let i = 0; i < longitude.length; ++i) {
    for (let j = 0; j < latitude.length; ++j) {
      const index = longitude.length * j + i;
      const value: number = values[index];

      const y = latitude[j];
      let x = longitude[i];
      if (x > 180) {
        x -= 360;
      }
      const trueValue =
        value === fill || Number.isNaN(value) ? 0 : value * scale + offse

      if (trueValue && trueValue < min) {
        min = trueValue;
      }
      if (trueValue && trueValue > max) {
        max = trueValue;
      }

      result.push([x, y, trueValue]);
    }
  }

  return { data: result, variables: variableNames, min, max };
}
```

1.13 src/components/Side.tsx

```
import {
  Table,
  Button,
  Space,
  TableColumnsType,
  Layout,
```

```
  Select,
  DatePicker,
  Row,
  Col,
} from "antd";
import {
  ExclamationCircleTwoTone,
  LoadingOutlined,
  MoreOutlined,
} from "@ant-design/icons";
import { AddLayerButton } from "~/components/AddLayerButton";
import atom, { Layer } from "~/atoms/layer";
import { useAtom } from "jotai";
import ui from "~/atoms/ui";
import { datetime } from "~/atoms/datetime";
import { EditLayerButton } from "./EditLayerButton";

/**
 * Renders the side panel which contains the layer list and global contro
 * @returns The rendered side panel.
 */
export function Side() {
  const [date, setDate] = useAtom(datetime);
  const [layers, setLayers] = useAtom(atom.layers);
  const [projection, setProjection] = useAtom(ui.projection);
  const columns: TableColumnsType<Layer> = [
    {
      title: "Layer",
      key: "name",
      dataIndex: "name",
      width: "100%",
      render: (value, layer) => {
        if (
          layer.dataset &&
          layer.dataset.variables &&
          layer.dataset.variables.length > 1
        ) {
          const variables = layer.dataset.variables;
```

```
          return (
            <Select
              style={{ width: "100%", margin: "-4px -12px" }}
              defaultValue={variables[0]}
              value={layer.variable || variables[0]}
              disabled={!layer.visible}
              onChange={(variable) => {
                setLayers({
                  action: "edit",
                  layer: {
                    key: layer.key,
                    variable: variable === variables[0] ? undefined : var
                  },
                });
              }}
              options={variables.map((variable) => ({
                label: `${value} / ${variable}`,
                value: variable,
              }))}
              bordered={false}
            />
          );
        }

        return value;
      },
    },
    {
      title: <Button type="text" size="small" icon={<MoreOutlined />} />,
      align: "center",
      key: "actions",
      render: (layer) => <EditLayerButton layer={layer} />,
    },
  ];

  return (
    <Layout.Sider width={350} style={{ padding: "8px", background: "#fff"
      <Space direction="vertical" size="large" style={{ width: "100%" }}>
        <Row>
```

```
      <Col span={24}>
        <Table
          style={{ width: "100%" }}
          size="small"
          columns={columns}
          dataSource={layers}
          footer={AddLayerButton}
          rowKey="key"
          rowSelection={{
            type: "checkbox",
            onSelect: (layer, selected) => {
              setLayers({
                action: "edit",
                layer: { key: layer.key, visible: selected },
              });
            },
            selectedRowKeys: layers
              .filter((layer) => layer.visible)
              .map((layer) => layer.key),
            renderCell: (_checked, layer, _index, _origin) =>
              (layer.state === "loading" && (
                <LoadingOutlined
                  style={{ width: "16px", color: "#1677ff" }}
                />
              )) ||
              (layer.state === "error" && (
                <ExclamationCircleTwoTone
                  twoToneColor="#fadb14"
                  style={{ width: "16px" }}
                />
              )) ||
              _origin,
          }}
          pagination={false}
          showHeader={false}
        />
      </Col>
    </Row>
    <Row gutter={8} align="middle">
```

```
    <Col span={6}>Projection</Col>
    <Col span={18}>
      <Select
        style={{ width: "100%" }}
        value={projection}
        onChange={(projection) => {
          setProjection(projection);
          location.reload();
        }}
        options={[
          {
            label: "Mercator",
            value: "mercator",
          },
          {
            label: "Globe",
            value: "globe",
          },
        ]}
      />
    </Col>
  </Row>
  <Row gutter={8} align="middle">
    <Col span={6}>Date</Col>
    <Col span={11}>
      <DatePicker
        style={{ width: "100%" }}
        value={date}
        onChange={(value) => {
          if (value) {
            setDate(value);
          }
        }}
      />
    </Col>
    <Col span={7}>
      <Select
        defaultValue="0"
        style={{ width: "100%" }}
```

```
                value={date.get("hour").toString()}
                onChange={(value) => {
                  setDate(date.startOf("day").set("hour", parseInt(value)))
                }}
                options={[
                  { value: "0", label: "00:00" },
                  { value: "1", label: "01:00" },
                  { value: "2", label: "02:00" },
                  { value: "3", label: "03:00" },
                  { value: "4", label: "04:00" },
                  { value: "5", label: "05:00" },
                  { value: "6", label: "06:00" },
                  { value: "7", label: "07:00" },
                  { value: "8", label: "08:00" },
                  { value: "9", label: "09:00" },
                  { value: "10", label: "10:00" },
                  { value: "11", label: "11:00" },
                  { value: "12", label: "12:00" },
                  { value: "13", label: "13:00" },
                  { value: "14", label: "14:00" },
                  { value: "15", label: "15:00" },
                  { value: "16", label: "16:00" },
                  { value: "17", label: "17:00" },
                  { value: "18", label: "18:00" },
                  { value: "19", label: "19:00" },
                  { value: "20", label: "20:00" },
                  { value: "21", label: "21:00" },
                  { value: "22", label: "22:00" },
                  { value: "23", label: "23:00" },
                ]}
              />
            </Col>
          </Row>
        </Space>
      </Layout.Sider>
    );
}
```

1.14 src/components/Palette.tsx

```
import { Select, SelectProps } from "antd";
import { colors } from "~/utils/colors";

/**
 * Palette selector element.
 */
const palettes: SelectProps["options"] = Object.entries(colors).map(
  ([type, palettes]) => {
    return {
      label: type,
      options: Object.entries(palettes).map(([title, colors]) => {
        return {
          value: `${type}.${title}`,
          label: (
            <div
              style={{
                display: "flex",
                justifyContent: "space-between",
                width: "100%",
              }}
            >
              <div style={{ width: "50%" }}>{title}</div>
              <div
                style={{
                  width: "50%",
                  display: "flex",
                  overflow: "hidden",
                  borderRadius: "2px",
                  margin: "4px 0",
                }}
              >
                {colors.map((color, index) => (
                  <div
                    key={`${type}.${title}.${index}`}
                    style={{ background: color, width: "100%" }}
                  />
                ))}
              </div>
```

```
            </div>
          ),
        };
      }),
    };
  }
);

/**
 * Renders a palette selector.
 * @param props The props to pass to the `Select` component.
 * @returns The rendered palette selector.
 */
export function Palette(props: Omit<SelectProps, "options">) {
  return <Select {...props} options={palettes} />;
}
```

1.15 src/components/Map.tsx

```
import { useMemo } from "react";
import { useAtomValue } from "jotai";
import DeckGL from "@deck.gl/react/typed";
import { _GlobeView, COORDINATE_SYSTEM } from "@deck.gl/core/typed";
import { BitmapLayer } from "@deck.gl/layers/typed";
import { TileLayer } from "@deck.gl/geo-layers/typed";

import layer from "~/atoms/layer";
import ui from "~/atoms/ui";

/**
 * Tile (OSM) layer for the map.
 * @see https://deck.gl/docs/api-reference/core/tile-layer
 */
const tileLayer = new TileLayer({
  data: [
    "https://a.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}@2x.png",
    "https://b.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}@2x.png",
    "https://c.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}@2x.png",
  ],
```

```
  tileSize: 256,
  maxRequests: 20,
  minZoom: 0,
  maxZoom: 16,
  zoomOffset: 1,
  renderSubLayers: (props) => {
    const {
      boundingBox: [min, max],
    } = props.tile;

    return [
      new BitmapLayer(props, {
        data: undefined,
        image: props.data,
        bounds: [...min, ...max] as [number, number, number, number],
        _imageCoordinateSystem: COORDINATE_SYSTEM.CARTESIAN,
      }),
    ];
  },
});

/**
 * The initial view state of the map.
 * @see https://deck.gl/docs/api-reference/core/deck#initialviewstate
 */
const initialViewState = {
  longitude: 37.618423,
  latitude: 55.751244,
  zoom: 1,
};

/**
 * Renders the map.
 * @returns The rendered component.
 * @see https://deck.gl/docs/api-reference/core/deck
 */
export function Map() {
  const layers = useAtomValue(layer.layers);
  const projection = useAtomValue(ui.projection);
```

```
  const deckGlLayers = useMemo(
    () => layers.map(({ layer }) => layer).filter(Boolean),
    [layers]
  );

  const view = useMemo(() => {
    return projection === "globe"
      ? new _GlobeView({ id: "globe", controller: true, resolution: 10 })
      : undefined;
  }, [projection]);

  return (
    <DeckGL
      views={view}
      initialViewState={initialViewState}
      style={{ position: "relative" }}
      layers={[tileLayer, ...deckGlLayers]}
      getTooltip={(data) => {
        const { object } = data;
        return (
          (object &&
            typeof object.value === "number" &&
            `${Math.round(object.value)}`) ||
          null
        );
      }}
      controller={true}
    />
  );
}
```

1.16 src/components/Layout.tsx

```
import { Layout as AntLayout } from "antd";

import { Side } from "~/components/Side";
import { Layer } from "~/components/Layer";
import { Map } from "~/components/Map";
```

```
/**
 * Renders the application layout.
 */
export function Layout() {
  return (
    <AntLayout style={{ minHeight: "100vh" }}>
      <Layer />
      <Side />
      <AntLayout.Content>
        <Map />
      </AntLayout.Content>
    </AntLayout>
  );
}
```

1.17 src/components/Layer.tsx

```
import {
  Button,
  Drawer,
  Space,
  Form,
  TreeSelect,
  Input,
  Select,
  Slider,
} from "antd";
import { useAtom, useSetAtom } from "jotai";
import { edit, layers, LayerSettings } from "~/atoms/layer";
import { Palette } from "~/components/Palette";

import reanalysis_era5_land from "datasets/reanalysis-era5-land.json";
import goes_16 from "datasets/goes-16.json";
import { DeleteOutlined } from "@ant-design/icons";
import { useEffect } from "react";

const tree = [...reanalysis_era5_land, ...goes_16];
```

```
/**
 * Renders a drawer to edit a layer.
 * @returns The rendered component.
 */
export function Layer() {
  const [layer, setLayer] = useAtom(edit);
  const [form] = Form.useForm<LayerSettings>();
  const opacity = Form.useWatch("opacity", form);
  const setLayers = useSetAtom(layers);

  const cancel = () => {
    setLayer(null);
  };

  useEffect(() => {
    if (layer) {
      form.setFieldsValue(layer);
    }
  }, [layer]);

  const remove = () => {
    if (layer && layer.key) {
      setLayers({ action: "remove", layer: { key: layer.key } });
    }
    setLayer(null);
  };

  const onFinish = (settings: LayerSettings) => {
    if (layer && layer.key) {
      setLayers({ action: "edit", layer: { ...layer, ...settings } });
    } else {
      setLayers({
        action: "add",
        layer: { ...layer, ...settings, key: `layer${Date.now()}` },
      });
    }
    cancel();
  };
```

```
const selectProduct = (_value: string, labels: React.ReactNode[]) => {
  if (!form.isFieldTouched("name")) {
    form.setFieldValue("name", labels[0] as string);
  }
};

return (
  <Drawer
    title={layer && layer.key ? "Edit layer" : "Add layer"}
    width={640}
    afterOpenChange={(open) => {
      if (!open) {
        form.resetFields();
      }
    }}
    extra={
      <Space>
        <Button type="dashed" onClick={cancel}>
          Cancel
        </Button>
        {layer && layer.name ? (
          <Button
            danger
            type="dashed"
            icon={<DeleteOutlined />}
            onClick={remove}
          >
            Remove layer
          </Button>
        ) : null}
        <Button type="primary" onClick={form.submit}>
          Save
        </Button>
      </Space>
    }
    open={Boolean(layer)}
    onClose={cancel}
    placement="left"
  >
```

```
<Form
  layout="vertical"
  name="edit-layer"
  form={form}
  onFinish={onFinish}
  initialValues={{
    type: "h3",
    palette: "Divergent.RdYlGn",
    opacity: 0.5,
    visible: true,
    ...layer,
  }}
>
  <Form.Item name="key" hidden>
    <Input />
  </Form.Item>
  <Form.Item name="product" label="Product" rules={[{ required: tru
    <TreeSelect
      treeData={tree}
      onChange={selectProduct}
      treeNodeLabelProp="label"
      treeExpandAction="click"
      showSearch
    />
  </Form.Item>
  <Form.Item name="name" label="Layer name" rules={[{ required: tru
    <Input />
  </Form.Item>
  <Form.Item name="palette" label="Colors" rules={[{ required: true
    <Palette />
  </Form.Item>
  <Form.Item name="type" label="Type" rules={[{ required: true }]}>
    <Select
      options={[
        { label: "Grid", value: "grid" },
        { label: "Hexagon", value: "h3" },
        { label: "Contour", value: "contour" },
      ]}
    />
```

```
        </Form.Item>

        <Form.Item label="Opacity" rules={[{ required: true }]}>
          <Form.Item noStyle name="opacity">
            <Input type="number" name="Opacity" min={0} max={1} step={0.1
          </Form.Item>
          <Slider
            value={opacity}
            onChange={(value) => form.setFieldValue("opacity", value)}
            step={0.01}
            min={0}
            max={1}
          />
        </Form.Item>
      </Form>
    </Drawer>
  );
}
```

1.18 src/components/EditLayerButton.tsx

```
import { MoreOutlined } from "@ant-design/icons";
import { Button } from "antd";
import { useSetAtom } from "jotai";
import { Layer, edit } from "~/atoms/layer";

/**
 * Renders a button that, when clicked, sets the `edit` atom to the value
 * @param props.layer The layer to edit.
 * @returns The rendered button.
 */
export function EditLayerButton({ layer }: { layer: Layer }) {
  const setEditLayer = useSetAtom(edit);

  return (
    <Button
      type="text"
      size="small"
      icon={<MoreOutlined />}
```

```
      onClick={() => {
        const {
          name,
          key,
          product,
          type,
          palette,
          opacity,
          visible,
          year,
          month,
          day,
          time,
        } = layer;
        setEditLayer({
          name,
          key,
          product,
          type,
          palette,
          opacity,
          visible,
          year,
          month,
          day,
          time,
        });
      }}
    />
  );
}
```

1.19 src/components/AddLayerButton.tsx

```
import { Button } from "antd";
import { useSetAtom } from "jotai";
import { PlusOutlined } from "@ant-design/icons";

import * as atom from "~/atoms/layer";
```

```
/**
 * Renders a button to add a new layer.
 * @returns The rendered component.
 */
export function AddLayerButton() {
  const setEditLayer = useSetAtom(atom.edit);

  return (
    <Button
      icon={<PlusOutlined />}
      onClick={() => setEditLayer({ visible: true })}
      type="primary"
      style={{ width: "100%" }}
    >
      Add layer
    </Button>
  );
}
```

1.20 src/atoms/ui.ts

```
import { atomWithStorage } from "jotai/utils";

/**
 * Projection of the map
 */
export const projection = atomWithStorage<"mercator" | "globe">(
  "projection",
  "globe"
);

export default { projection };
```

1.21 src/atoms/layer.ts

```
import { atom } from "jotai";
import { atomFamily, atomWithStorage, loadable } from "jotai/utils";
```

```ts
import type { Layer as DeckGLLayer } from "@deck.gl/core/typed";

import { Dataset, datasets } from "~/atoms/dataset";
import { datetime } from "~/atoms/datetime";
import { getDeckGlLayer, getParams } from "~/utils/layer";

/**
 * Layer settings
 */
export type LayerSettings = {
  key: string; // unique key
  name: string; // display name
  product: string; // product name
  type: "h3" | "grid" | "contour" | "raw"; // layer type
  palette: string; // palette name
  opacity: number; // opacity
  visible: boolean; // visibility

  variable?: string; // variable of the selected product
  year?: string; // override global year
  month?: string; // override global month
  day?: string; // override global day
  time?: string; // override global time
};

/**
 * Layer instance
 */
export type Layer = LayerSettings & {
  state: "loading" | "loaded" | "error";
  layer?: DeckGLLayer;
  dataset?: Dataset;
};

/**
 * Partial layer settings
 */
type PartialSettings = Partial<LayerSettings> & Pick<LayerSettings, "key"
```

```
/**
 * Action to initialize layer settings on startup
 */
export type Init = {
  action: "init";
};

/**
 * Action to add a new layer
 */
export type Add = {
  action: "add";
  layer: LayerSettings;
};

/**
 * Action to edit a layer
 */
export type Edit = {
  action: "edit";
  layer: Partial<LayerSettings> & Pick<LayerSettings, "key">;
};

/**
 * Action to remove a layer
 */
export type Remove = {
  action: "remove";
  layer: Partial<LayerSettings> & Pick<LayerSettings, "key">;
};

/**
 * Layer settings update action payload
 */
export type Update = Init | Add | Edit | Remove;

/**
 * Layer settings stored in local storage
 */
```

```
const settings = atomWithStorage<LayerSettings[]>("layers", [
  {
    visible: true,
    key: "layer0",
    product: "reanalysis-era5-land/2m_temperature",
    name: "Temperature Contour",
    palette: "Constant.White",
    type: "contour",
    opacity: 1,
  },
  {
    visible: true,
    key: "layer1",
    product: "reanalysis-era5-land/2m_temperature",
    name: "Temperature Fill",
    palette: "Divergent.RdYlGn",
    type: "h3",
    opacity: 0.75,
  },
]);

/**
 * Layer settings dictionary
 * @param name - layer name
 * @returns layer settings
 */
const family = atomFamily((_name: Layer["key"]) => {
  const layerSettings = atom<PartialSettings | null>(null);
  return atom(
    (get) => {
      const settings = get(layerSettings);
      const date = get(datetime);
      const params = settings && getParams(settings, date);
      const dataset = params && get(loadable(datasets(params)));

      if (!settings) {
        return null;
      }
```

```
    if (!params || !dataset) {
      return {
        ...settings,
        state: "error",
      };
    }

    if (dataset.state === "loading") {
      return {
        ...settings,
        state: "loading",
      };
    }

    if (dataset.state === "hasError") {
      console.warn(dataset.error);
      return {
        ...settings,
        state: "error",
      };
    }

    if (
      dataset.state === "hasData" &&
      dataset.data != null &&
      settings.palette != null &&
      settings.opacity != null &&
      settings.visible != null
    ) {
      return {
        ...settings,
        state: "loaded",
        dataset: dataset.data,
        layer: getDeckGlLayer(dataset.data, settings as LayerSettings),
      };
    }
  },
  (_get, set, layer: PartialSettings | null) => {
    set(layerSettings, layer);
```

```
    }
  );
});

/**
 * Layer settings being edited or created
 */
export const edit = atom<Partial<LayerSettings> | null>(null);

/**
 * Layer settings array atom
 */
export const layers = atom(
  (get) =>
    get(settings)
      .map(({ key }) => get(family(key)))
      .filter((layer): layer is Layer => layer != null),
  (get, set, update: Update) => {
    if (update.action === "init") {
      for (const layer of get(settings)) {
        set(family(layer.key), layer);
      }
    }

    if (update.action === "add") {
      set(family(update.layer.key), update.layer);
      set(settings, (settings) => [...settings, update.layer]);
    }

    if (update.action === "edit") {
      const layer = get(settings).find(
        (layer) => layer.key === update.layer.key
      );
      const updatedLayer = { ...layer, ...update.layer };
      set(family(update.layer.key), updatedLayer);
      set(settings, (settings) =>
        settings.map((layer) => {
          if (layer.key !== update.layer.key) {
            return layer;
```

```
          }
          return {
            ...layer,
            ...update.layer,
          };
        })
      );
    }

    if (update.action === "remove") {
      set(family(update.layer.key), null);
      set(settings, (settings) =>
        settings.filter((layer) => layer.key !== update.layer.key)
      );
    }
  }
);

layers.onMount = (setLayers) => {
  setLayers({ action: "init" });
};

export default { edit, layers };
```

1.22 src/atoms/datetime.ts

```
import dayjs, { Dayjs } from "dayjs";
import { atom } from "jotai";
import { atomWithStorage } from "jotai/utils";

const _datetime = atomWithStorage("date", "2023-04-01T00:00:00Z");

/**
 * Global date and time
 */
export const datetime = atom(
  (get) => dayjs(get(_datetime)),
  (_get, set, date: Dayjs) => set(_datetime, date.toISOString())
);
```

1.23 src/atoms/dataset.ts

```ts
import * as idb from "idb-keyval";
import { atom } from "jotai";
import { atomFamily } from "jotai/utils";
import { load } from "~/worker";
import { BinaryFeatures } from "@loaders.gl/schema";

/**
 * Dataset parameters to load
 */
export type DatasetParams = {
  key: string; // unique key
  path: string; // path to the dataset
  payload?: Record<string, string | number>; // payload to pass to the lo
  source: "GOES-16" | "ERA5"; // source of the dataset
  type: "h3" | "grid" | "contour" | "raw"; // layer type
  variable?: string; // variable of the selected product
};

export type DatasetResult = {
  key: string; // unique key
  min: number; // min value
  max: number; // max value
  date: Date; // date
  variables?: string[]; // available variables
  availableDates?: Date[]; // available dates
} & (
  | {
      type: "h3" | "grid" | "raw"; // layer type
      key: string; // unique key
      count: number; // number of pixels
      buffer: ArrayBuffer | SharedArrayBuffer; // buffer to store the dat
    }
  | {
      type: "contour"; // layer type
      features: BinaryFeatures; // flat GeoJSON features
    }
```

```
);

/**
 * Dataset instance
 */
export type Dataset = DatasetParams & DatasetResult;

const MAX_BUFFER_SIZE = 256 * 1024 * 1024; // 256 MB

/**
 * Dataset atom family (dictionary)
 */
export const datasets = atomFamily(
  (params: DatasetParams) => {
    return atom(async () => {
      const cached = (await idb.get(params.key)) as Dataset;
      if (cached) {
        return cached;
      }
      const buffer = new SharedArrayBuffer(MAX_BUFFER_SIZE);
      const data = await load({ ...params, buffer });

      return data;
    });
  },
  (a, b) => a.key === b.key
);
```