cambecc / earth Public

a project to visualize global weather conditions

earth.nullschool.net

MIT license

4.7k stars 1k forks

Star Watch

Code Issues 82 Pull requests 3 Actions Projects Wiki Security Insigh

master

cambecc Merge pull request #48 from paulirish/patch-1 … on Sep 2, 2016 175

View code

README.md

earth

NOTE: the location of `dev-server.js` has changed from `{repository}/server/` to `{repository}/`

"earth" is a project to visualize global weather conditions.

A customized instance of "earth" is available at http://earth.nullschool.net.

"earth" is a personal project I've used to learn javascript and browser programming, and is based on the earlier Tokyo Wind Map project. Feedback and contributions are welcome! ...especially those that clarify accepted best practices.

building and launching

After installing node.js and npm, clone "earth" and install dependencies:

```
git clone https://github.com/cambecc/earth
cd earth
npm install
```

Next, launch the development web server:

```
node dev-server.js 8080
```

Finally, point your browser to:

```
http://localhost:8080
```

The server acts as a stand-in for static S3 bucket hosting and so contains almost no server-side logic. It serves all files located in the `earth/public` directory. See `public/index.html` and `public/libs/earth/*.js` for the main entry points. Data files are located in the `public/data` directory, and there is one sample weather layer located at `data/weather/current`.

*For Ubuntu, Mint, and elementary OS, use `nodejs` instead of `node` instead due to a naming conflict.

getting map data

Map data is provided by Natural Earth but must be converted to TopoJSON format. We make use of a couple different map scales: a simplified, larger scale for animation and a more detailed, smaller scale for static display. After installing GDAL and TopoJSON (see here), the following commands build these files:

```
curl "http://www.nacis.org/naturalearth/50m/physical/ne_50m_coastline.zip" -o
ne_50m_coastline.zip
curl "http://www.nacis.org/naturalearth/50m/physical/ne_50m_lakes.zip" -o ne_50m_lakes.zip
curl "http://www.nacis.org/naturalearth/110m/physical/ne_110m_coastline.zip" -o
ne_110m_coastline.zip
curl "http://www.nacis.org/naturalearth/110m/physical/ne_110m_lakes.zip" -o
ne_110m_lakes.zip
unzip -o ne_\*.zip
ogr2ogr -f GeoJSON coastline_50m.json ne_50m_coastline.shp
ogr2ogr -f GeoJSON coastline_110m.json ne_110m_coastline.shp
ogr2ogr -f GeoJSON -where "scalerank < 4" lakes_50m.json ne_50m_lakes.shp
ogr2ogr -f GeoJSON -where "scalerank < 2 AND admin='admin-0'" lakes_110m.json
ne_110m_lakes.shp
ogr2ogr -f GeoJSON -simplify 1 coastline_tiny.json ne_110m_coastline.shp
ogr2ogr -f GeoJSON -simplify 1 -where "scalerank < 2 AND admin='admin-0'" lakes_tiny.json
ne_110m_lakes.shp
topojson -o earth-topo.json coastline_50m.json coastline_110m.json lakes_50m.json
lakes_110m.json
topojson -o earth-topo-mobile.json coastline_110m.json coastline_tiny.json lakes_110m.json
lakes_tiny.json
cp earth-topo*.json <earth-git-repository>/public/data/
```

getting weather data

Weather data is produced by the Global Forecast System (GFS), operated by the US National Weather Service. Forecasts are produced four times daily and made available for download from NOMADS. The files are in GRIB2 format and contain over 300 records. We need only a few of these records to visualize wind data at a particular isobar. The following commands download the 1000 hPa wind vectors and convert them to JSON format using the grib2json utility:

```
YYYYMMDD=<a date, for example: 20140101>
curl "http://nomads.ncep.noaa.gov/cgi-bin/filter_gfs.pl?
file=gfs.t00z.pgrb2.1p00.f000&lev_10_m_above_ground=on&var_UGRD=on&var_VGRD=on&dir=%2Fgfs.${YY
 -o gfs.t00z.pgrb2.1p00.f000
grib2json -d -n -o current-wind-surface-level-gfs-1.0.json gfs.t00z.pgrb2.1p00.f000
cp current-wind-surface-level-gfs-1.0.json <earth-git-
repository>/public/data/weather/current
```

font subsetting

This project uses M+ FONTS. To reduce download size, a subset font is constructed out of the unique characters utilized by the site. See the `earth/server/font/findChars.js` script for details. Font subsetting is performed by the M+Web FONTS Subsetter, and the resulting font is placed in `earth/public/styles`.

Mono Social Icons Font is used for scalable, social networking icons. This can be subsetted using Font Squirrel's WebFont Generator.

implementation notes

Building this project required solutions to some interesting problems. Here are a few:

- The GFS grid has a resolution of 1°. Intermediate points are interpolated in the browser using bilinear interpolation. This operation is quite costly.
- Each type of projection warps and distorts the earth in a particular way, and the degree of distortion must be calculated for each point (x, y) to ensure wind particle paths are rendered correctly. For example, imagine looking at a globe where a wind particle is moving north from the equator. If the particle starts from the center, it will trace a path straight up. However, if the particle starts from the globe's edge, it will trace a path that curves toward the pole. Finite difference approximations are used to estimate this distortion during the interpolation process.
- The SVG map of the earth is overlaid with an HTML5 Canvas, where the animation is drawn. Another HTML5 Canvas sits on top and displays the colored overlay. Both canvases must know where the boundaries of the globe are rendered by the SVG engine, but this pixel-for-pixel information is difficult to obtain directly from the SVG elements. To workaround this problem, the globe's bounding sphere is re-rendered to a detached Canvas element, and the Canvas' pixels operate as a mask to distinguish points that lie outside and inside the globe's bounds.

- Most configuration options are persisted in the hash fragment to allow deep linking and back-button navigation. I use a backbone.js Model to represent the configuration. Changes to the model persist to the hash fragment (and vice versa) and trigger "change" events which flow to other components.
- Components use backbone.js Events to trigger changes in other downstream components. For example, downloading a new layer produces a new grid, which triggers reinterpolation, which in turn triggers a new particle animator. Events flow through the page without much coordination, sometimes causing visual artifacts that (usually) quickly disappear.
- There's gotta be a better way to do this. Any ideas?

inspiration

The awesome hint.fm wind map and D3.js visualization library provided the main inspiration for this project.

Releases

No releases published

Packages

No packages published

Contributors 2

cambecc Cameron Beccario

paulirish Paul Irish

Languages

● **JavaScript** 83.5% ● **HTML** 14.4% ● **CSS** 2.1%