Geographic data analysis and geo web applications with R and ReactJS

Dr Layik Hama[*]*— University of Leeds, Leeds Institute for Data Analytics (LIDA)*
Dr Robin Lovelace[†]*— University of Leeds, Institute for Transport Studies (ITS) and Leeds Institute for Data Analytics (LIDA)*

2019-01-21

Summary

Data scientists and R users in general do not have to rely on other programming languages to deploy their projects on the web. There are R packages that enable us to combine the best of front end technology such as ReactJS with web mapping libraries like LeafletJS to create geospatial web applications.

Keywords: R package, JavaScript, ReactJS, web mapping

1 Introduction

Scripting languages (Mazzoni et al. 2006) have been powering the web for a long time. The most widely used (thanks to content management systems such as Wordpress and giants like Facebook) is perhaps PHP (Gilmore and Bryla 2007). But PHP zhao2012hiphop is not used for data science or geospatial data processing. R language is. When we embarked on a project focused on web mapping and data processing and analysis, we faced a dilemma many data scientits and web application developers in general would face: do we use R for the end web app or as an intermediate step? We found out that there had already been some work in this regard inspired by Flask (Grinberg 2018) in Python called `plumber` (Trestle Technology, LLC 2018). We realised, we could actually use `plumber` to power our next project. The packager `plumber` is built on another R package called `httpuv` which is built "on top of the libuv and http-parser C libraries, both of which were developed by Joyent, Inc" (Joe Cheng 2018). RStudio (RStudio Team 2015) has a similar product called Shiny(Beeley 2016). Aside from the freemium license, `geoplumber` takes a more standard web application framework approach and uses a more standard frontend choice (ReactJS) compared to Shiny.

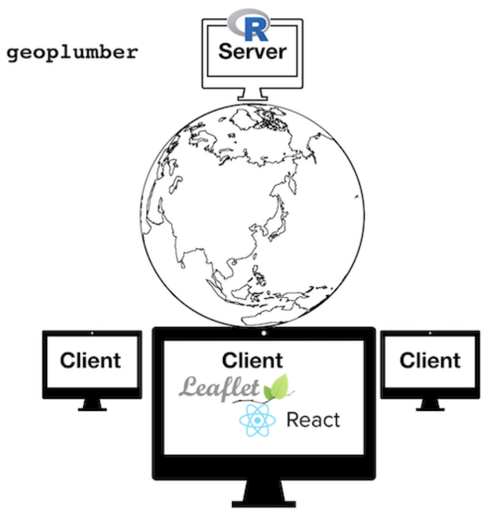


Figure 1: Running R as backend and ReactJS + LeafletJS at the frontend of a web application.

2 Approach: Combining technologies for data science and the web

Antoher stack at our disposal was Facebook's ReactJS (Fedosejev 2015). The scalability of ReactJS (powers Facebook) and its programming pattern are two of its separate strengths. These, along with our skills in using

[*]l.hama@leeds.ac.uk
[†]r.lovelace@leeds.ac.uk

it which made it the main candidate to be combined with `plumber`. Suddenly, we found ourselves generalizing what we had done for the project to become an R package that could be used by other developers.

A separate set of technologies that are powered by JavaScript but is a world of its own, is web mapping and technologies such as LeafletJS and Mapbox JS. Since the emergence of WebGL and adoption of it by the likes of Mapbox JS, we could see how we could combine these tech stacks to build a modern "web application framework" that could be used to analyse and visualize large geospatial datasets.

Because our focus is on geospatial data analysis and visualization, the result is an R package called `geoplumber`. The package is a combination of these technologies which are loosely coupled and could be used for non-geospatial datasets. This loose coupling is done with attention, in order to allow developers who are used to tools provided for ReactJS development, for example Facebook provided tools could do their work isolated from the R ecosystem. The tools are based on `npm` (Node Package Manager) or other package managers for NodeJS, the one used in `geoplumber` is called Create-React-App (CRA) NodeJS package.

The package has been developed with consideration for data scientits who would be analysing geospatial data interactively. There are functions to support such tasks and similar to other mapping and visualization packages developed for the same purpose with using more "static"

3 Use case: a web app for visualising road traffic casualties

To develop a web API (Application Programming Interface) and a web front end to consume data served from the API, backend R and browser markup is needed. The `geoplumber` package includes development functions to create web apps, add ReactJS scripts, add R powered plumber API end-points and of course required R functions for building and running the app.

A geoplumber app, as it stands, is a standard `npm` package generated by CRA. For the API, an `R` directory containing a `plumber.R` file is added, which is used by the underlying `plumber` package. To create an app:

```r
library(geoplumber)
gp_create("/tmp/geoplumber")
```

The directory and files structure of a `geoplumber` applications looks like this:

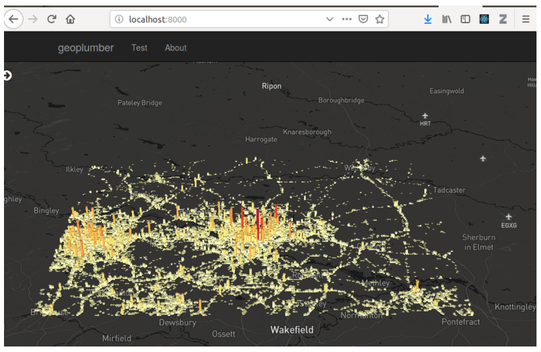
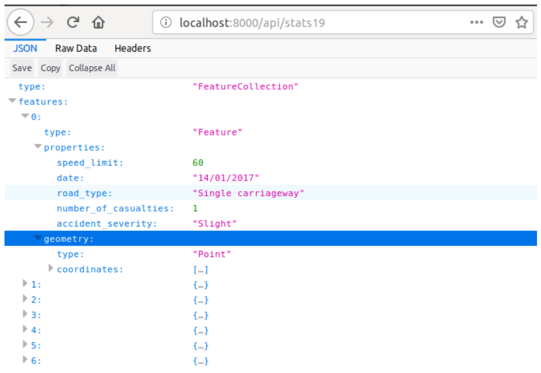
```
+- R/plumber.R       # backend code
+- README.md
+- package.json      # npm package file
+- public            # public facing docs
+- src               # frontend JS code.
```

We can then do all our data processing straight from R and serve the data using API end-points. `plumber` works by adding tags in front of standard R functions. For example to geneate an end-point that returns an R object which contains JSON data in a parameter called `accidents_geojson`, we could write a function like this:

```r
#' @get /api/stats19
all_geojson <- function(res){
  res$headers$`Content-type` <- "application/json"
  res$body <- accidents_geojson
  res
}
```

The first line which is a standard R comment line, has an `@get` tag which means next end-point is going to be a HTTP GET path. The function has a `response` object which can be modified and returned. In this case, we set the HTTP response `content-type` and also load the body of the response object with the JSON obect to be returned. You could guess that this would then be accessible from the Flask like tags `@get /api/stats19`, translated into paths like `localhost:8000/api/stats19`.

In our example stats19(R. Lovelace et al. 2019) R package is used for data processing. Using stats19 we can get the crashes for years since 1979 from DfT (Department for Transport, United Kingdom). We have also worked on the front end. We use the functions in geoplumber to work on the development and finally deploy our application. A screen shot of an example is shown in Figure 2.

Frontend view	The API serving the data
	

localhost:8000/api/stats19

```
type:                       "FeatureCollection"
features:
  0:
    type:                   "Feature"
    properties:
      speed_limit:          60
      date:                 "14/01/2017"
      road_type:            "Single carriageway"
      number_of_casualties: 1
      accident_severity:    "Slight"
    geometry:
      type:                 "Point"
      coordinates:          […]
  1:                        {…}
  2:                        {…}
  3:                        {…}
  4:                        {…}
  5:                        {…}
  6:                        {…}
```

Table 1: A geoplumber app for road casualty data. On the left, the frontend is made up of Uber's DeckGL built on top of Mapbox JS. On the right the same data served from `/api/stats19` end-point.

4 Geospatial Databases

Although currently the work in progress repository[1] does not include one, it is possible to add a database of choice to the stack. Due the light weight Flask/plumber type of API frameworks, it is possible to make use of the full potential of R language. For example, to connect to a MySQL database running on a Linux machine, with username and password defined at the users `~/.my.cnf` file as per MySQL convenctions. We can then create API end-points that can connect to a MySQL instance with a `geoplumber` schema defined in it, using `RMySQL` and `DBI` packages as follows:

```
con <- DBI::dbConnect(RMySQL::MySQL(), group = "my-db")
# we can send SQL queries such as selecting a schema
DBI::dbGetQuery(conn = con, "use geoplumber;")
#> data frame with 0 columns and 0 rows
DBI::dbListTables(con)
#> character(0)
```

Add the output of the above into another end-point:

```
#' @get /api/tables
tables <- function(res){
  res$headers$`Content-type` <- "application/json"
  res$body <- tables_list_geojson
  res
}
```

[1]Repository link: https://github.com/ATFutures/geoplumber

5 Deployment

Standard deployment documentation[2] for `plumber` application is provided by the developers. In our small scale deployment experience, we have deployed the geoplumber apps using Docker virtualization technology as a reverse proxy using a standard Nginx HTTP server.

6 Acknowldgements

The work carried out is funded by Consumer Data Resarch Center (CDRC) at Leeds Institute for Data Analytics (LIDA) at University of Leeds. The open source ecosystem enables projects like Linux and can benefit geoplumber immensely.

References

Beeley, Chris. 2016. *Web Application Development with R Using Shiny.* Packt Publishing Ltd.

Fedosejev, Artemij. 2015. *React. Js Essentials.* Packt Publishing Ltd.

Gilmore, W Jason, and Bob Bryla. 2007. *Beginning Php and Oracle: From Novice to Professional.* Apress.

Grinberg, Miguel. 2018. *Flask Web Development: Developing Web Applications with Python.* " O'Reilly Media, Inc."

Joe Cheng, Jeroen Ooms, Hector Corrada Bravo. 2018. *Httpuv: HTTP and Websocket Server Library* (version 1.4.5.1). https://cran.r-project.org/web/packages/httpuv/index.html.

Lovelace, Robin, R Lovelace, M Morgan, L Hama, and M Padgham. 2019. "Stats19: A Package for Working with Open Road Crash Data." *Journal of Open Source Software.* doi:10.21105/joss.01181.

Mazzoni, Silvia, Frank McKenna, Michael H Scott, Gregory L Fenves, and others. 2006. "OpenSees Command Language Manual." *Pacific Earthquake Engineering Research (PEER) Center* 264.

RStudio Team. 2015. *RStudio: Integrated Development Environment for R.* Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Trestle Technology, LLC. 2018. *Plumber: An Api Generator for R* (version 0.4.6). https://cran.r-project.org/web/packages/plumber/index.html.

[2]see https://www.rplumber.io/docs/hosting.html