NodeGIS: A Container-based Web GIS Application Development Tool

Mateus Queiroz Cunha
Information Systems Laboratory
Federal University of Campina Grande
Campina Grande, Brazil
email: mateusqueiroz@copin.ufcg.edu.br

Cláudio de Souza Baptista
Information Systems Laboratory
Federal University of Campina Grande
Campina Grande, Brazil
email: baptista@computacao.ufcg.edu.br

***Abstract*—Nowadays, several tools which foster the development of Web Geographical Information Systems (GIS) have emerged, associated with the outspread of spatial information. Although there are many solutions, most of them are complex, requiring specific skills from users. Therefore, there exists a demand for a solution that simplifies the process of developing and deploying Web GIS applications. This paper presents NodeGIS, an open-source tool that provides a graphical interface for developing Web GIS applications without code writing or complex server configuration. NodeGIS enables the user to plot vector maps, to perform overlay and customization operations, zooming, panning, tooltip, and spatial and non-spatial queries. NodeGIS uses a container-based and Representational State Transfer (REST) architecture, thus facilitating the deployment of Web GIS applications. NodeGIS can also be used to teach GIS, requiring no software installation from students.**

Keywords*—*GIS; Web GIS; containerization; REST.

I. Introduction

Along with the ubiquity of spatial information in current applications, several commercial and free tools have emerged, as well as Application Programming Interfaces (APIs) that promote the development of Web Geographic Information System (GIS) applications. Large companies in the GIS sector have provided Web GIS solutions, such as Microsoft Location Technologies - Azure Maps [1], Nokia Here SDK [2], Amazon AWS Location API [3], Google Maps API [4], deck.gl [5], Apple Maps API [6], mapbox [7], Uber API [8], ESRI ArcGIS API [9], QGIS [10], MapServer [11], GeoServer [12], among many other solutions. Also, many database management systems provide native support for spatial data, such as the relational: PostgreSQL/PostGIS, MySQL, SQL Server, Oracle and IBM DB2; NewSQL: SAP Hana, CockRoachDB and SingleStore; and NoSQL: MongoDB, CouchDB, Cassandra, among others.

Concerning desktop GIS, there are many commercial and free solutions, such as QGIS and gvSIG [13]. However, when we look at Web GIS solutions [14] there are numerous solutions using map servers, spatial database servers and frontends. These three tiers contain different complexity and requirements, many times requiring specific skills from the user in the different technologies [15]–[19]. A server-only application that requires specialized frontend development to consume and display data, a severe learning curve before a user can create a useful application, and high prices for commercial (and even educational) purposes are a few examples of encountered challenges. Among Web GIS well known solutions, we mention GeoServer, MapServer, QGIS Server [20], and ArcGIS Server. Therefore, there is a need for a tool that simplifies the process of developing and publishing a Web GIS application [21]. That is the main purpose of the NodeGIS tool.

NodeGIS is an open-source tool that provides a graphical interface for developing a Web GIS without the need to develop code or configure servers. The NodeGIS application deployment is done via Docker [22] containers using two images, one for the frontend and the other one for the backend, in addition to a PostgreSQL/PostGIS container, making the application ready to use with geographic data.

Our contribution concerns to bring the low-effort experience of developing desktop GIS applications to the Web environment. In addition, the NodeGIS tool may be used in the teaching and learning process of geoprocessing techniques, providing both the teacher and students with a tool that allows them to graphically and interactively explore various GIS resources, without needing to install software, using only a browser with an internet connection.

In the remainder of this paper, Section II describes some of the related work, followed by Section III, which presents the architecture and main features of NodeGIS. Section IV describes how to create a Web GIS using NodeGIS. Later, Section V focuses on a user evaluation of the developed solution. Final considerations are the subject of Section VI.

II. Related Work

The evolution of Web Mapping applications in recent years has highlighted the demand for geographic applications that fulfill user requirements [23]. GIS applications have been impacted by the technological advancements of the Web and distributed systems, however, there is a technological gap between Web-oriented information systems and existing GIS solutions [24].

Recent research has focused on the importance of the user interface when working with GIS applications [25]. There are also research on GIS teaching dealing with the positive impact of bringing a desktop GIS experience to a Web GIS application [26][27]. This can also help non-expert users to be able to create Web GIS applications for their respective areas of expertise.

More recently, several Web GIS solutions have been proposed specifically for a single domain, such as landslide and flood mapping [28], spatial accessibility of urban medical facilities [29], seismic vulnerability of old urban areas [30], marine spatial planning [31], and promotion of archaeological and environmental tourism [32]. These research works addressed domain-specific challenges associated with the complexity of using GIS tools.

III. THE NODEGIS SOLUTION

Among the main features of NodeGIS, we highlight the addition of vector layers, conventional and spatial queries; customization of map layers; construction of thematic maps; tooltip; interactive query on the map, where it is possible to carry out complex spatial queries in an easy way through the application's interface, with no need for SQL knowledge; search for features on the map; use of multiple spatial databases; data tables, as well as filters and selections that can be applied to the data and visualized on the map.

Figure 1 presents an example of a NodeGIS running application that contemplates the following layers: map of the municipalities, railways and highways of the Brazilian state of Paraíba. In addition, Figure 1 also depicts a spatial query with a buffer operation of the central railway in the state, highlighted in green.

The architectural design of NodeGIS is based on a Representational State Transfer (REST) architecture, segmented into the frontend and backend, associated with one or more spatial relational databases, as detailed in Figure 2. We detail each architectural aspect of the NodeGIS tool in the following subsections.

A. Frontend

The NodeGIS frontend consists of a React [33] application, which uses an implementation of the Flux [34] frontend data flow pattern with four main elements: the View, represented by the React components that display the application data to the user; the Store, responsible for obtaining, manipulating and storing data in the application; the Dispatcher, responsible for managing the data flow, distributing actions coming from the View to their respective Store(s); and, finally, the Action, which are functions that carry data to the Stores derived from user actions. Figure 3 shows the details of the frontend architecture and its components communication.

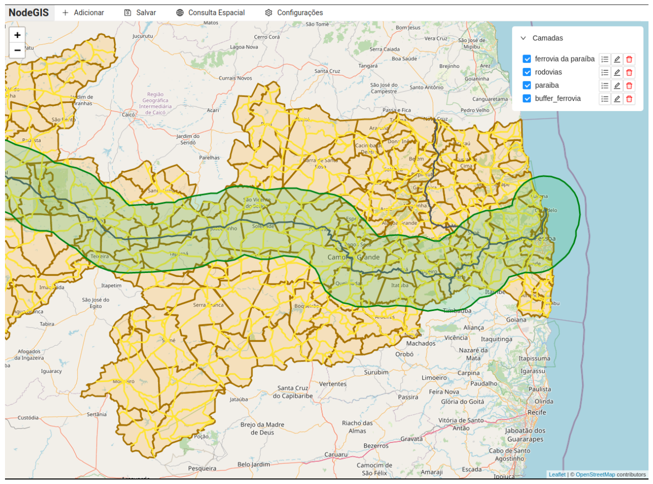


Figure 1. An application example deployed in NodeGIS.

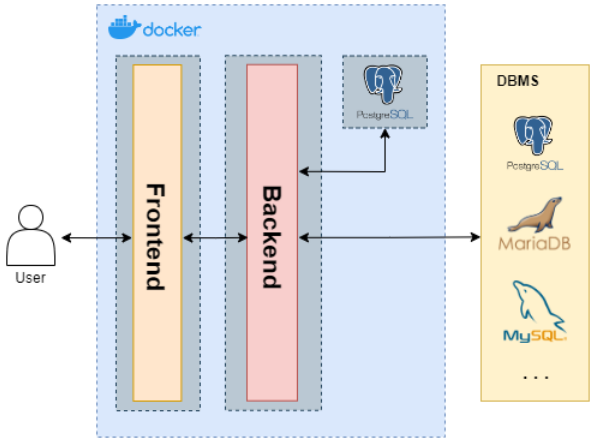


Figure 2. NodeGIS architecture.

The main element of the frontend is the map, developed using the Leaflet [35] library, which has a specific version for React applications. All the data on the map comes from their respective Store, which stores data obtained using HTTP requests to the backend. The backend is responsible for fetching geographic data from the database and converting it to GeoJSON, a geospatial data format that uses JavaScript Object Notation (JSON). The Leaflet library can read the data in a layer on the map directly from GeoJSON format, not requiring any adjustment or additional structuring of the geographic data.

When users want to add a new vector layer to the map, NodeGIS displays a list of all database tables containing spatial data, as well as the names of the spatial columns for each table. Users can also define the data displayed in the tooltip of map features, layer styling, and the layer name on the map using the frontend interface.

A JavaScript object represents each layer, which are stored

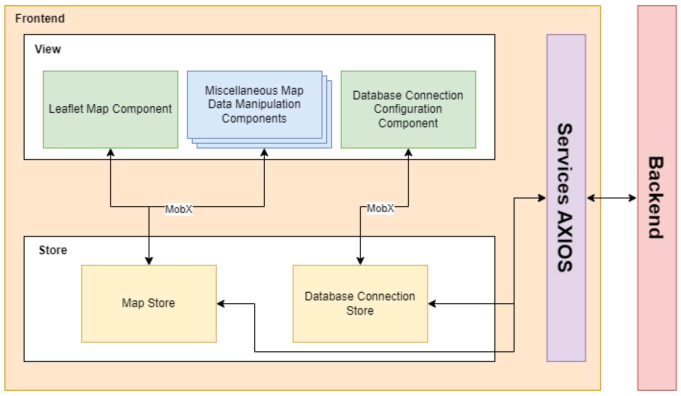

Frontend
View
Leaflet Map Component
Miscellaneous Map Data Manipulation Components
Database Connection Configuration Component
MobX
MobX
Store
Map Store
Database Connection Store
Services AXIOS
Backend

Figure 3. NodeGIS frontend architecture.

in the Map Store. All layer objects follow a well-defined pattern, having the following attributes:

- *name*: name given to the layer.
- *key*: database table that represents the layer.
- *geometryColumn*: data type of the geometry column used in the layer, which can be Polygon, LineString or Point.
- *styles*: an object containing the layer's static style definitions (if the styleType attribute is equal to static).
- *styleType*: an attribute that defines the type of layer styling, whether it is static (static) or thematic (choropleth).
- *displayColumns*: list of objects containing columns that will have their values displayed at the tooltip of the layer features on the map.
- *choroplethStyleDfinition*: an object containing style definitions for thematic maps (if the styleType is equal to choropleth).
- *data*: GeoJSON data returned by querying the database and received as a response to the HTTP request made to the backend.

Leaflet React builds each component using the data from the Map Store, rendering each element on the map. The component called MapContainer defines the external context of the map used in the application and renders the other components corresponding to each element added to the map. Each rendered element uses the data attribute of a vector layer object (described above), sending it to the GeoJSON component. However, we used the CircleMarker component for Point data to emphasize it and enable styling, given that the default markers, displayed when using the GeoJSON component, cannot be customized. That would cause confusion when displaying multiple layers on the map.

Users can also perform spatial querying, an essential feature for a Web GIS application. It can be done either as native SQL queries or by selecting features on the map. The SQL spatial query redirects the query entered manually by the user to the backend and, later, to the database. As for the query which uses feature selection on the map, it englobes the selection of individual features or entire layers according to the chosen spatial operation.

Spatial queries have their structure initially built on the frontend, using a query definition object. Next, the backend receives that object via an HTTP request. This query is then built and subsequently sent to the database.

B. Backend

The NodeGIS backend consists of a web server capable of responding to HTTP requests, following a REST standard, using NodeJS in association with the express framework. As shown in Figure 4, the NodeGIS backend architecture is composed of four segments:

- *Routes*: consists of an access layer that redirects HTTP requests to their respective controllers.
- *Controller*: this layer deals with requests made to the API and their responses based on the data coming from the repository after processing.
- *Repositories*: responsible for building spatial and general queries made to the database.
- *Database Clients*: responsible for the connection to the database(s).

In order to provide versatility and possibility of choice to the user, NodeGIS has compatibility with different spatial relational Database Management Systems (DBMS), namely: PostgreSQL, MySQL, MariaDB, SQLite (with its SpatiaLite extension) and CockroachDB. We implemented a specific database client for each supported DBMS, extending the existing NodeJS database client and enabling NodeGIS to deal with any divergences in SQL implementations. In addition to the external databases used with the application, the backend has also an instance of SQLite running internally. This instance assists in managing the application data, storing which layers the users recorded in their visualization, and storing connection data to the other databases used.

Most of the repositories defined in the backend redirect API calls to the database using basic queries, except for the queryRepository, that is responsible for building the spatial query requests from the frontend. The supported spatial operations are: union, difference, intersection, contains, crosses, touches, disjunction, intersects, buffer, centroid, area, distance, length and perimeter.

When building the spatial query, the frontend makes an HTTP request to the backend containing a body in the format shown in Figure 5. This object comprises the following attributes: a data parameter of the spatial query; the second data parameter of the spatial query; and operation, a string representing the spatial operation to be applied. The second property can be empty for queries that do not apply an operation between different features (or sets of features). For example, that is the case for operations such as buffer and area.

In more detail, the *first* and *second* properties store the tables used in the query as keys of the property object. Each table key also stores properties, which are the *data* and the *geometryColumn*. The *data* property holds a list of unique identifiers of the table features, called Global Identifiers (GIDs). The query uses features whose GID values are in the list — an empty list means considering the entire table. The *geometryColumn* property stores which spatial column of the

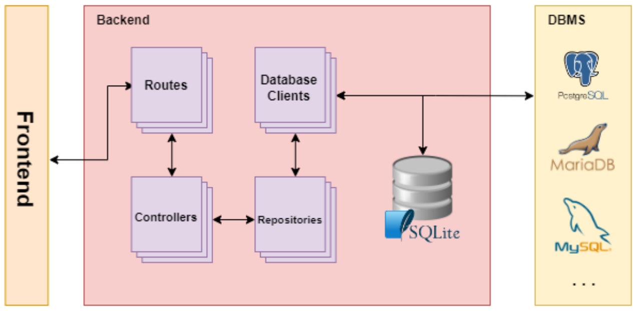


Figure 4. Backend architecture details.

table will be used in the spatial query.

For each of the spatial query parameters, a subquery is built to obtain the requested data, being simply a union of the requested features (if there are feature GIDs in the data property) or the entire table (if the data property is empty). Then, from the subqueries, the spatial function is applied according to the string referenced in the operation property, using the spatial columns of each subquery as parameters for the spatial function.

At the end of the build process, the *ST_AsGeoJSON* function wraps the query to convert the geometry data into the GeoJSON format, making it easier to manipulate, read and represent the data on the map.

C. Containerization

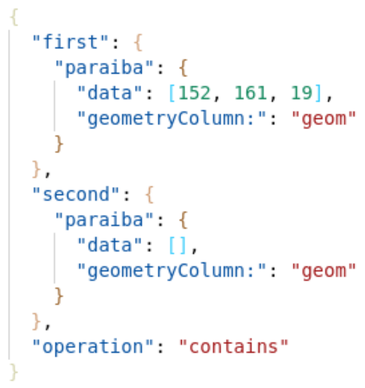
In order to facilitate the Web GIS application deployment, Docker images are available containing the required modules to run the application (frontend and backend). The images are publicly accessible on DockerHub [36].

To facilitate the execution of the Docker images into containers, therefore, the application deployment, two scripts were written in the root directory of the application GitHub repository [37]. One of the scripts runs the application in a local environment (*run-application-local.sh*), and the other in a production environment with public access (*run-application-prod.sh*). These scripts have already each container parameters and environment variables correctly configured. However, a script change is possible according to specific infrastructure needs.

In order to provide a ready-to-use environment, we have a third Docker image of a PostgreSQL database with PostGIS containing sample data. This containerized database is configured as the default and added to the application startup settings.

IV. Creating a Web GIS application using NodeGIS

NodeGIS aimed at providing an application startup that reduces the impacts of integration between frontend, backend and databases, as well as influences of the environment where this application will be used (operating system, for example).



```
{
  "first": {
    "paraiba": {
      "data": [152, 161, 19],
      "geometryColumn:": "geom"
    }
  },
  "second": {
    "paraiba": {
      "data": [],
      "geometryColumn:": "geom"
    }
  },
  "operation": "contains"
}
```

Figure 5. JSON format sent to the backend when doing spatial queries.

Containerization is fundamental in this process. Hence, installing Docker is the only prerequisite for running NodeGIS.

A. Initial Configuration

As described in section III-C, shell scripts were made available in the root directory of the application repository to speed up the application initialization. The scripts are *run-application-local.sh* (local environment) and *run-application-prod.sh* (production environment). Both perform this configuration, however, with different parameters and purposes.

When executing the desired script (e.g., using the "*bash run-application-local.sh*" command), three Docker images on the host machine will be used (and fetched from DockerHub, if not existent locally): the frontend, the backend and the PostgreSQL spatial database. Finally, the three containers will be initialized with their respective parameters, showing an output that contains the hash code of each initialized Docker container and the access URL for the application.

The execution of the scripts is similar in both environments. However, when NodeGIS runs with the production script, it depends on the configuration of the host machine so that external users can access it. The environment must have a public IP and ports 8080 and 8081 unlocked. The *stop-application.sh* script is responsible for stopping NodeGIS execution, using the names assigned to the containers in the execution script.

All script parameters may be modified according to the user needs. An example of this scenario is when a user already has a database containing spatial data, where the initialization of the PostgreSQL container becomes optional, or when the ports used in the production environment need to be changed.

With the application fully initialized, the user can use the features of NodeGIS through the top menu. For example, the feature to add vector layers to the map can be found in the "Add" menu, followed by the "Vector Layer" option. Users can select which database table to use and its spatial column. In addition, users can customize the information displayed in the features' tooltip and the layer style.

B. Spatial Querying

Spatial queries are based on map feature selection and can be accessed in the menu using the "Spatial Query" selector, followed by the "From Selection" option. This feature allows the user to select the desired operation and the parameters needed to carry it out (features or layers). Users can execute queries resulting in geometries, such as the buffer operation. Note that users can change the style of the map feature for better visualization. It is also possible to record the result of spatial queries in new tables in the database, thus possibly reusing them in new queries or saving them as the default visualization of NodeGIS.

Hence, users can easily and quickly: initialize NodeGIS, add vector layers to the map, perform spatial queries and save visualizations generated from existing tables or queries obtained from the NodeGIS.

V. NodeGIS User Evaluation

In order to assess user experience with NodeGIS, we interviewed twenty users using a questionnaire. All users have some experience on information system development. We adapted the Post-Study System Usability Questionnaire (PSSUQ) [38] to have only relevant questions according to a suitable NodeGIS scenario.

The final objective of applying this questionnaire is to obtain metrics regarding the satisfaction of different users with the NodeGIS tool. The questionnaire contains eight statements, which users must classify according to their respective experiences when using the tool. Here are the statements used:

1) I have some Geographic Information Systems (GIS) knowledge/experience.
2) The system was simple to use.
3) The system has all the features and capabilities I expected.
4) I was able to complete tasks and scenarios quickly using the system.
5) The system gave me error messages that clearly told me how to solve the issues.
6) The information (such as online help, on-screen messages and other documentation) provided by the system was clear.
7) The organization of information on the screen was clear.
8) I would recommend the system to others.

Each interviewee evaluated each statement with a score from 1 to 7, where 1 means strongly disagree and 7 strongly agree. When starting the evaluation process, users had access to a descriptive video [39] containing the main features of NodeGIS. In addition, two simple activities were passed on to the interviewees: adding a vector layer to the map and performing a spatial query with the added layer. Then, the use of the tool was unrestricted and unsupervised. All users accessed the same NodeGIS deployment and geographic database.

Observing Figure 6, which contains the result of Question 1, we can see that a little more than 50% of the interviewees declared themselves knowledgeable of GIS techniques, which provides us with a good balance of people who have already had contact with other GIS tools and those who have not.

Figure 7 depicts the remainder of the evaluation results. About 75% of the evaluations of each statement had a score greater than or equal to 5, representing an excellent result. In addition, score 7 was most common in the evaluations of each statement. It also appeared tied to scores 6 or 5.

The high occurrence of scores close to 7 reflects the general average of the evaluation obtained with the questionnaire, which was 6.27 (disregarding the first statement, given that it does not represent the user's experience directly with NodeGIS). The closer this general average is to 7, the greater the user satisfaction when using NodeGIS.

VI. Conclusion

The development of Web GIS applications has increased exponentially. This paper presents The NodeGIS Web GIS application development tool that is based on Docker containers

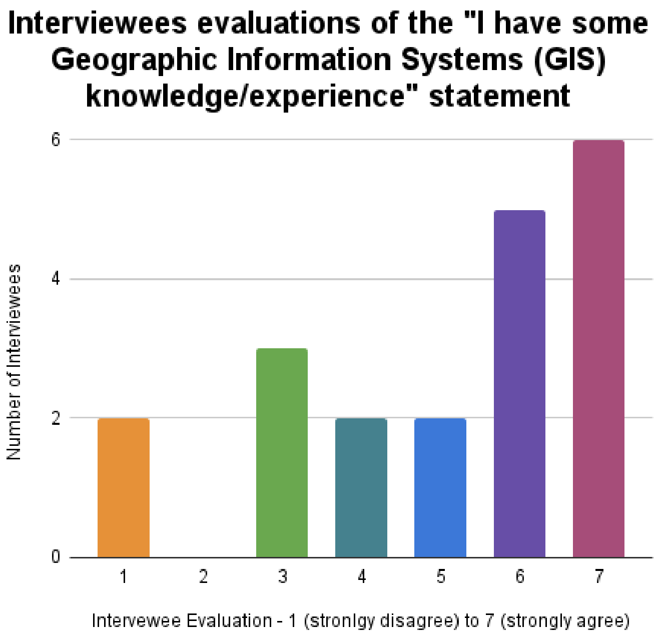


Figure 6. Interviewees evaluation of the "I have some Geographic Information Systems (GIS) knowledge/experience" statement.

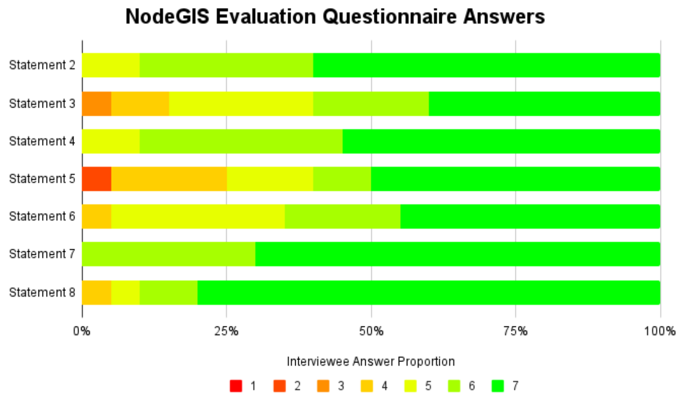


Figure 7. NodeGIS Evaluation Questionnaire Answers.

to simplify deployment. NodeGIS allows plotting vector maps, *overlay* and *layer customization*, *zooming*, *panning*, *tooltip*, performing queries on conventional attributes and various spatial operations on data. NodeGIS also targets aid in teaching GIS without requiring students to install any software.

A NodeGIS user can develop a web GIS application using a simple configuration that requires no specific distributed computation skills. Hence, users may focus on their target domain instead of technical aspects. Even though the NodeGIS tool is designed to be simple, expert users can still use their skills to create unique and advanced spatial queries and use different spatial database management systems.

One of the challenges encountered was the dynamic rendering and data management of the vector layers in the frontend, which can result in a complex implementation when dealing with "pure" JavaScript libraries. The React version of the Leaflet library was essential to overcome this problem, contributing to an implementation that is easy to maintain and evolve.

In future work, we intend to aggregate new simple features to the NodeGIS platform such as table editing, data exportation, and multimedia data association (e.g., associating images and videos with map features). Furthermore, we also plan to include new complex features such as 3D visualizations, point cloud and raster support.

ACKNOWLEDGEMENT

The second author would like to thank the National Council for Scientific and Technological Development (CNPq), Brazil, for partially funding this research.

REFERENCES

[1] "Azure Maps - Geospatial Mapping APIs." Microsoft Azure. https://azure.microsoft.com/en-us/services/azure-maps/ (accessed Mar. 3, 2023).
[2] "Build apps with HERE Maps API and SDK Platform Access." Here Developer. https://developer.here.com/ (accessed Mar. 3, 2023).
[3] "Amazon Location Service." Amazon Web Services. https://aws.amazon.com/location/ (accessed Mar. 3, 2023).
[4] "Google Maps Platform." Google Developers. https://developers.google.com/maps (accessed Mar. 3, 2023).
[5] "deck.gl." deck.gl. https://deck.gl/ (accessed Mar. 3, 2023).
[6] "Apple Maps." Apple Developer. https://developer.apple.com/maps/ (accessed Mar. 3, 2023).
[7] "Maps, geocoding, and navigation APIs & SDKs." Mapbox. https://mapbox.com/ (accessed Mar. 3, 2023).
[8] "Developers." Uber. https://developer.uber.com/ (accessed Mar. 3, 2023).
[9] "ArcGIS Developers." ArcGIS. https://developers.arcgis.com/ (accessed Mar. 3, 2023).
[10] "Welcome to the QGIS project." QGIS. https://www.qgis.org/ (accessed Mar. 3, 2023).
[11] "Welcome to MapServer." MapServer documentation. https://mapserver.org/ (accessed Mar. 3, 2023).
[12] "GeoServer." GeoServer. http://geoserver.org/ (accessed Mar. 3, 2023).
[13] "gvSIG Desktop." Portal gvSIG. http://www.gvsig.com/en/products/gvsig-desktop (accessed Mar. 3, 2023).
[14] D. Moretz, "Internet GIS," in *Encyclopedia of GIS*, Springer International Publishing, 2016, pp. 1–7.
[15] D. Yu and J. Yin, "Internet GIS and system dynamic modeling in urban public safety and security studies: A conceptual framework," in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 207–216.
[16] A. Milosavikevic, "An application framework for rapid development for web based GIS : Ginis Web," in *Geospatial Services and Applications for the Internet*, Springer US, 2008, pp. 48–71.
[17] A. Paiva, E. da Silva, F. Leite, and C. de Souza Baptista, "A multiresolution approach for internet GIS applications," in *Proceedings. 15th International Workshop on Database and Expert Systems Applications, 2004.*, IEEE, 2004, pp. 809–813.
[18] F. Yin and M. Feng, "A webgis framework for vector geospatial data sharing based on open source projects," in *Proceedings. The 2009 International Symposium on Web Information Systems and Applications (WISA 2009)*, Academy Publisher, 2009, pp. 124–127.
[19] E. Nash, P. Korduan, S. Abele, and G. Hobona, "Design requirements for an ajax and web-service based generic internet GIS client," in *11th AGILE International Conference on Geographic Information Science*, University of Girona, Spain, 2008, pp. 1–6.
[20] "QGIS Server GuideManual." QGIS Documentation. https://docs.qgis.org/3.22/en/docs/server_manual/ (accessed Mar. 3, 2023).
[21] T. E. Chow, "The potential of maps APIs for internet GIS applications," *Transactions in GIS*, vol. 12, no. 2, pp. 179–191, Apr. 2008.
[22] "Docker: Accelerated, Containerized Application Development." Docker. https://www.docker.com/ (accessed Mar. 3, 2023).
[23] B. Veenendaal, M. A. Brovelli, and S. Li, "Review of web mapping: Eras, trends and directions," *ISPRS International Journal of Geo-Information*, vol. 6, no. 10, p. 317, Oct. 2017.
[24] A. Rowland, E. Folmer, and W. Beek, "Towards self-service GIS—combining the best of the semantic web and web GIS," *ISPRS International Journal of Geo-Information*, vol. 9, no. 12, p. 753, Dec. 2020.
[25] R. Unrau, A. Kudekar, and C. Kray, "Interaction pattern analysis for scpWebGIS/scp usability evaluation," *Transactions in GIS*, vol. 26, no. 8, pp. 3374–3388, Nov. 2022.
[26] C. Mah, D. Hong, V. Chen, and E. Stefanakis, "First-year engineering students' research experience in web mapping," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 55, no. 1, pp. 53–62, Mar. 2020.
[27] X. Xiang and Y. Liu, "Exploring and enhancing spatial thinking skills: Learning differences of university students within a web-based GIS mapping environment," *British Journal of Educational Technology*, vol. 50, no. 4, pp. 1865–1881, Sep. 2018.
[28] C. Iadanza *et al.*, "IdroGEO: A collaborative web mapping application based on REST API services and open data on landslides and floods in italy," *ISPRS International Journal of Geo-Information*, vol. 10, no. 2, p. 89, Feb. 2021.
[29] J. Wang *et al.*, "Assessing the spatial accessibility of urban medical facilities in multi-level and multi-period scales based on web mapping API and an improved potential model," *ISPRS International Journal of Geo-Information*, vol. 11, no. 11, p. 545, Oct. 2022.
[30] C. Columbro, R. R. Eudave, T. M. Ferreira, P. B. Lourenço, and G. Fabbrocino, "On the use of web mapping platforms to support the seismic vulnerability assessment of old urban areas," *Remote Sensing*, vol. 14, no. 6, p. 1424, Mar. 2022.
[31] A. González, C. Kelly, and A. Rymszewicz, "Advancements in web-mapping tools for land use and marine spatial planning," *Transactions in GIS*, vol. 24, no. 2, pp. 253–267, Dec. 2019.
[32] M. Luppichini *et al.*, "Web mapping and real–virtual itineraries to promote feasible archaeological and environmental tourism in versilia (italy)," *ISPRS International Journal of Geo-Information*, vol. 11, no. 9, p. 460, Aug. 2022.
[33] "React - A JavaScript library for building user interfaces." React. https://reactjs.org/ (accessed Mar. 3, 2023).
[34] "Flux." Flux. https://facebook.github.io/flux/ (accessed Mar. 3, 2023).
[35] "Leaflet - a JavaScript library for interactive maps." Leaflet. https://leafletjs.com/ (accessed Mar. 3, 2023).
[36] M. Cunha, "mateusqc's Profile." Docker Hub. https://hub.docker.com/u/mateusqc (accessed Mar. 3, 2023).
[37] M. Cunha, "mateusqc/node-gis: An application to support the construction of Web GIS." GitHub. https://github.com/mateusqc/node-gis (accessed Mar. 3, 2023).
[38] J. Lewis, "Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ," vol. 2, Jan. 1992, pp. 1259–1263.
[39] M. Cunha, "NodeGIS - Main Features." YouTube. https://youtu.be/SZYesxzrBIc (accessed Mar. 3, 2023).