Cartography M.Sc.

Master thesis

Vector tile cache connecting effective spatial communication and geospatial AI

Sharon Chawanji

Technical University of Munich

TECHNISCHE UNIVERSITÄT WIEN Vienna University of Technology

TECHNISCHE UNIVERSITÄT DRESDEN

UNIVERSITY OF TWENTE.

2020

MASTERARBEIT

Vector tile cache connecting effective spatial communication and geospatial AI

eingereicht von

Sharon Chawanji

Matrikelnummer 01234567

12 Oktober 2020

Ausgeführt am Department für Geodäsie und Geoinformation
der Technischen Universität Wien

unter der Anleitung
von

Dipl.-Ing. Dr. Markus Jobst, TU Wien

Univ.Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner, TU Wien

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

MASTER'S THESIS

Vector tile cache connecting effective spatial communication and geospatial AI

Submitted by

Sharon Chawanji

Matriculation number 01234567

12 October 2020

Conducted at the Department of Geodesy and Geoinformation

Vienna University of Technology

Under the supervision

of

Dipl.-Ing. Dr. Markus Jobst, TU Wien

Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner, TU Wien

Signature (Author)

Signature (Supervisor)

Vector tile cache connecting effective spatial communication and geospatial AI

submitted for the academic degree of Master of Science (M.Sc.)
conducted at the Department of Aerospace and Geodesy
Technical University of Munich

Author:	Sharon, Chawanji
Study course:	Cartography M.Sc.
Supervisor:	Dipl.-Ing. Dr. Markus Jobst, TU Wien
Reviewer:	M.Sc. Mathias Gröbe, TU Dresden
Chair of the Thesis Assessment Board:	Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner, TU Wien

Date of submission: 12.10.2020

Statement of Authorship

Herewith I declare that I am the sole author of the submitted master's thesis entitled:

"Vector tile cache connecting effective spatial communication and geospatial AI"

I have fully referenced the ideas and work of others, whether published or unpublished. Literal or analogous citations are clearly marked as such.

Munich, 12.10.2020 Sharon, Chawanji

Acknowledgements

I would like to express my sincere gratitude to all who continually supported and motivated me during my thesis research and writing process. To mention a few my main advisor Dipl.-Ing. Dr. Markus Jobst, Prof. Georg Gartner and Mathias Grobe who expertly guided me with their immense knowledge. I would also like to thank the rest of my thesis committee including Juliane Cron, Paulo Raposo and Wangshu Wang for their tremendous effort to motivate me and the help with organisation. Not forgetting my fellow Cartorebels of the Cartography Master 8th intake and my friends and family in Zimbabwe. I thank you all.

Abstract

The growth of the Web has been characterised by publication and establishment of linked open data on the Web powered by technologies such as RDF and SPARQL. Much of this data are embedded with geographic content that links them to a position on Earth's surface. This data can be extracted from sources such as social media, remote sensing and government portals. The abundance of the data on the Web create a huge pool of spatial big data. By employing geospatial artificial intelligence (geoAI), knowledge can be extracted from the spatial big data that can answer questions to real world phenomena. Since the data is geolinked it can be integrated with geospatially enabled Web services so that it can be visualised on Web GIS applications.

As most datasets on the Web are distributed on different computers connected by the internet, Web services are developed with the purpose of providing interoperable data access, data integration and data processing functionalities. One such Web service is the Open Geospatial Consortium (OGC) Table Joining Service (TJS). TJS provides an interface that can take a geospatial framework on one node, and attribute data on another node and merge them based on common geographic identifiers.

The goal of this thesis is to examine the feasibility of implementing a TJS that uses cached vector tiles as the geospatial framework and Comma Separated Value (CSV) format for attribute data. TJS specification requires that attribute data be formatted in an XML based structure called GDAS. However, most datasets published on the Web are in CSV format. Additionally, RDF data stores can be queried via SPARQL endpoints, the results of these queries are in tabular format and can be converted to CSV data format.

To achieve the thesis goal a prototype implementation of the TJS concept is developed which ingest attribute data in CSV format and cached vector tiles. The two datasets are to be merged based on common geographic identifiers. Vector tiles are small pre-package containers of vector data. Vector tiles have several advantages over other means of distributing geographic features through the internet. Vector tiles are small, they can be cached for later use, they are rendered by the client and users are able to interact and use the underlying geographic features for further geographical processing and spatial analysis. The results of this thesis will be displayed as a Web thematic map in a simple OpenLayers Web map application.

Keywords: Table Joining Service, Vector tile cache, Linked Open Data, Geographic identifier, SPARQL

Abbreviations

API	Application Programming Interface
GDAS	Geolinked Data Access Service
GeoAI	Geospatial Artificial Intelligence
GIS	Geographic Information Systems
GLS	Geolinked Service
GSGF	Global Statistical Geospatial Framework
GWC	GeoWebCache
HTTP	Hyper Text Transfer Protocol
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LOD	Linked Open Data
OGC	Open Geospatial Consortium
OL	OpenLayers
OWS	Open Geospatial Consortium Web Services
RDF	Resource Descriptive Framework
SDGs	Sustainable Development Goals
SOA	Service Orientated Architecture
TJS	Table Joining Service
TMS	Tile Map Service
UN-GGIM	United Nations Committee of Experts on Global Geospatial Information Management
UNSC	United Nations Statistical Commission
WFS	Web Feature Service
WMS	Web Map Service
WMS-C	Web Map Service - Cache

WMTS	Web Map Tile Service

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Background and motivation

Spatial big data with high-precision and wide-coverage has exploded globally. Spatial big data include data from remotely sensed data, geo-social media data, and statistical data from business to government data (Jiang and Shekhar, 2017). This provides a good opportunity to enhance the national decision-making, social supervision, public services, and emergency capabilities (VoPham et al., 2018). According to Li et al., 2016 nearly 80% of data on the Web are embedded with geographic information that can be mapped and geovisualised (Li et al., 2016). Spatial big data can be made available as published structured data which is interlinked to other datasets also known ss Linked Data. Linked data technologies are vital building blocks to knowledge graphs and are the core of the Semantic Web. Knowledge graphs acquire concealed knowledge from an enormously large amount of interlinked data. This is achieved by integrating the interlinked data into an ontology and applying a semantic reasoner (Fensel et al., 2020). The Semantic Web, coupled with innovations in spatial science, artificial intelligence and high-performance computing form a scientific discipline called Geospatial artificial intelligence (geoAI) (VoPham et al., 2018).

The Semantic web makes it possible to develop technologies such as Resource Description Framework (RDF) and SPARQL that empower a Web of geolinked data to be used as knowledge bases. **"The Semantic Web refers to refers to W3C's vision of the Web of linked data with technologies that enable people to create data stores on the Web, build vocabularies and write rules for handling data"**[1]. RDF is a standard model for describing resources on the web [2]. RDF data can be accessed using a data exchange service called an SPARQL endpoint. SPARQL (an abbreviation for SPARQL Protocol and RDF Query Language) is a query language for RDF data, results from SPARQL queries resemble tabular data and can be converted to a CSV data format. Web Geographic Information Systems (GIS) technology enables the CSV data to be integrated with geographic frameworks based the embedded geographic information so that the data can be visualised on a Web based map.

For an extended period, the traditional desktop GIS has been used for joining attribute data, ,usually in tabular structure like CSV, with geographic data (Billen et al., 2006). Desktop GIS brings several limitation, including data sharing, it is usually limited to one hardware at a time and it requires users to be experts with months of training and experience (Abdalla and Esmail, 2018).

To date the development of the Web has made many to shift from desktop GIS to web GIS. Indeed, the Web GIS can be accessed by many users simultaneously with a significant abstraction of the technical background for novice users; it has a potential of offering more advanced full featured GIS enabled web services (Abdalla and Esmail, 2018). Vector tiling for example has been proved as the most effective way of delivering spatial data over the recent decade ("OGC

[1] https://www.w3.org/standards/semanticweb/
[2] https://www.w3.org/RDF/

Vector Tiles Pilot: WFS 3.0 Vector Tiles Extension Engineering Report," 2019). Vector tiles are containers of geographic data that have been broken down into tiles for easy transfer over the internet (Martinelli and Roth, 2015).

Most map service systems such as Google Maps are built on vector tile map models (Wan et al., 2016). Vector tiles have many advantages over other ways of delivering geospatial data over the internet because the tilling technology reduces the size of vector data into vector tile hence transferring and rendering of vector tiles more efficient and faster. Additionally, vector tiles are rendered by the client unlike the popularly used raster tiles which are rendered by the server (Burghardt et al., 2014). However, vector tile technology deals with a high access rates in order to provide fundamental information for geospatial processing and spatial analysis. Unlike the raster tiles which are created by breaking down a map image and are useful for displaying purposes, vector tiles allow users to interact with embedded geographic features.

To overcome the challenge of high access rates, caching mechanisms, like vector tile caches, are proposed to reduce the computational resource requirement and this increases the concurrent request rate a server can handle. Vector tile caches store data temporarily so that future requests of the data can be served faster. The data stored in a cache might be a product of earlier computation or a copy of data store (Netek et al., 2020). These characteristics make vector tiles highly flexible and optimised for an enhanced user experience. Additionally, the ability to utilise the raw vector data makes accessing, manipulating and styling of geographic features on the client side possible (Li et al., 2017). Ultimately, it gives an opportunity for vector tiles to be integrated with attribute datasets based on the embedded geographic identifiers in the vector tiles on the client side instead of the server side.

Existing studies on vector tiles have mostly focused on optimizing rendering of vector tile on web applications, efficient caching mechanisms for improved performance of Web map applications and improving vector tiles transmission rates (e.g., Antoniou et al., 2009, Gaffuri, 2012). However, vector tiles can also be used as a dataset capable of being integrated with other datasets based on common feature ID or any other feature attribute with unique geographic identifiers. The applicability of cached vector tiles as a source of geospatial data to be integrated with statistical data is not clearly understood.

1.2 Research Objectives

The aim of this thesis is to examine the potential of using cached vector tiles as a source of geospatial data for integration with statistical data based on Open Geospatial Consortium's Table Joining Service (TJS). Using cached vector tiles reduces the processing pressure on the server that would have otherwise been created by multiple requests (Shang, 2015). Multiple requests from the server at the same time can cause the server to slow down or shut down (Wan et al., 2016).

Table Joining Service is an open Web service standard that defines a way for geolinked data to be accessed from a remote location so that it can be joined with geographic data [3]. Geolinked data in the context of TJS refers to attribute data that can be linked to a geographic location. However, the attribute data itself does not carry the geometric

[3] http://www.opengeospatial.org/

information but rather a geographic identifier that can be linked to the corresponding geometry representing a geographic feature. This geographic identifier is also referred to as the framework key.

The TJS uses the principles of a Service Oriented Architecture (SOA), offering a potential solution for client-to-server based transformation and for merging geography and statistics with the benefits of interoperability and accessibility. A SOA satisfies some guiding principle given by the Global Statistical Geospatial Framework (GSGF) for integrating spatial and attribute data [4]. Specifically, principle number four which requires that data be published once and leaving it at its source. The data can be reused and accessed many times through Web services for merging geography and statistics.

The specific objectives of this research are to:

1. Examine the possibility of using cached vector tiles as a geospatial framework to be integrated with attribute data by means of a Table Joining Service.
2. Develop a prototype implementation to be used as a tool for integrating spatial data and attribute data.
3. Demonstrate through a simple Web map application the results of objective 1 using the prototype developed in objective 2.

1.3 Structure of the thesis

This research is organised as follows Section 2 gives the fundamentals and literatures review of the concept of TJS and vector tiling. Section 3 describes approaches, experimental data, softwares and technology used to achieve the research objectives. Section 4 describes the prototype implementation and Section 5 gives a discussion of the prototype implementation and Section 6 gives a conclusion and recommendation for future work.

[4] http://ggim.un.org/meetings/GGIM-committee/9th-Session/documents/The_GSGF.pdf

2. Fundamentals and literature review

2.1 Theoretical Background

Thematic maps are ranked amongst the most popular and important cartographic products for visualising spatial data. Being more intelligible and clearer, they present a less complicated presentation of thematic data, enabling users to connect information to a spatial location (Peterson, 2012, p. 143). Data is often collected according to spatial units such as country, district and municipality. Qualitative and/or quantitative thematic data can be displayed to show spatial distribution of the themes for the spatial units (Cammack, 2007). Thematic maps are used by experts such as geographers, researchers and urban planners as sources of information or they may be used to present results (Veldhuizen and Pfeffer, 2016). With the advancement in web technology different standards have been developed for geospatial webservices that can enable development of web thematic maps. (Veenendaal et al., 2017).

Web services are software applications with standardised programming interfaces; they make it possible for web applications to interact with each other over the internet (Alonso et al., 2013, p. 125). A web service functionality is exposed as a service with methods and a published interface a client can request for. A client is a software that accesses a web service from a server forming a client-server model. In many cases the server runs on a different computer thus a client accesses a web service through a computer network.

Since web services offer an interface for computer programs to communicate it is important that they are loosely coupled. This means software programs are deployed independent of each other and thus a program only needs to be executed as needed. This scenario is known as service oriented architecture (SOA) (Alonso et al., 2013, p. 131). In a SOA computer programs are in the form of services each program is separated into a distinct self-sufficient and network assessible component intended to solve a specific concern (Krafzig et al., 2005, p. 55). SOA is especially important for creating thematic maps in a clustered computer environment because many times computer hosting geospatial webservices and those hosting statistical data are remote.

The Open Geospatial Consortium (OGC) is the main authority in developing standard for Geoinformation technologies (GTI) web services (Peterson, 2012, p. 143). The OCG Web Services (OWS) provides interfaces for geospatial content and services, GIS processing and data discovery (Lupp, 2008). A complete description of OGC standards is available on the OGC website [5]. The following sections give an overview of standards that have previously been used and those that are still in use for developing and distributing Web thematic maps.

2.1.1 Web Map Service and Styled Layer Descriptor

The Web Mapping Service (WMS) standard is one of the earliest standards issued by OGC in 2002. WMS Implementation Specification provides an HTTP interface for requesting georeferenced map images from one or more

[5] https://www.ogc.org/docs/is

distributed databases [6]. WMS are created by a map server with data provided by a GIS database; thus, thematic maps can be created and saved as map images. Clients can request a desired map image of any size covering an arbitrary geographic bounding box (García et al., 2012). WMS are often used in combination with an OCG specification called Styled Layer Descriptor (SLD) (Bocher and Ertz, 2018). The SLD standard is used to describe the appearance of Web Map Services. Multiple customized styles can be published that can be used to style a WMS displaying thematic map.

WMS has many disadvantages even though it is still in use; it is based on a raster model. Raster models do not allow users access to the underlying geographic features; hence they can be used for displaying purposes only. A study by Cammack in 2007 gives an extensive review of uses and limitations of WMS in thematic mapping. In this study one of the major limitations of WMS is how it is tightly coupled to the underlying data; thus there is need to regenerate an entire WMS each time data geometry changes (Cammack, 2007, p. 446).

Another issue with WMS is that it not cacheable; thus, with each new request from a client map images have to be dynamically generated on the fly (García et al., 2012). Sometimes the data is of such high volume that each display upon each request is time consuming. Moreover, rendering of the WMS is done by the server which is computationally expensive especially when there are high requests for the same resource on the server. Consequently WMS overall performance and efficiency both on the client and server side is very poor (Davis et al., 2009).

The introduction of tilling schemes for WMS improved its ability to cache maps. Tilling schemes allow map images to be broken down into discrete image tiles giving birth to the OGC Web Map Tile Service Standard (WMTS) (Masó et al., 2010). This specification significantly improved performance because instead of computer applications dealing with the entire raster image, they only display the portion of image tiles requested by the client. Tile Map Service (TMS) serve as the basis of the WMTS where raster image is broken down into small raster tiles.

TMS is an open source specification for tiled web maps developed by Open Source Geospatial Foundation (OSGeo) [7]. TMS uses a tiling scheme where each tile in a zoom level is a parent to 4 tiles in the succeeding zoom levels (see Figure 1). In the case of WMTS raster tiles are generated by splitting a map image into tile-based grid. The position of each tile is based on the coordinates of the tile grid used to create it. To request a specific tile the request should be directed to the coordinate of the tile on the grid. The extraction format is in the form z/x/y.format where z is zoom level x is column position and y is row position. For example, to request for a tile in level 1 in column 1 and row 2 the format extraction would be 1/1/2.png.

Nevertheless, WMTS still suffers the same limitations as WMS in that they are both based on a raster model and interaction or manipulation with the geographic features is not possible.

[6] https://www.ogc.org/standards/wms
[7] https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

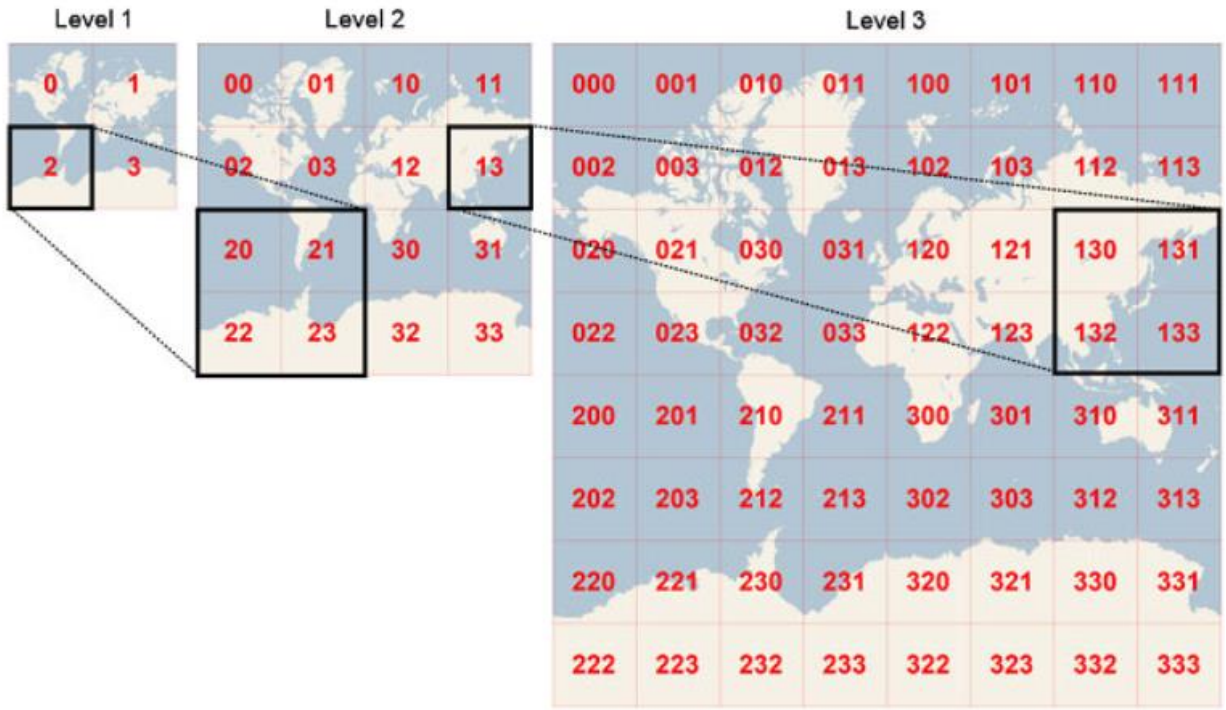

Figure 1. Vector tile scheme based on Quad tree structure in which each tile is a parent of 4 tiles in the subsequent zoom levels in a tile pyramid. Source (Balog and Houtmeyers, 2017)

2.1.2 Geolinking Service and Geolinked Data Access Service

Hong and Lin (2005) research was the earliest study in which a system was developed, that make use of OGC web services for joining attribute data with geospatial data to produce web thematic maps. The system architecture for their research was under the assumption that geospatial data and statistical data are hosted on different computers in a distributed computer network. Naturally there would be requirement for a Web Service to connect the two computers so that the data can be merged to create a Web thematic map. Geolinking Service (GLS) defines an interface for services that provide the ability to join datasets that contain thematic data about geographic features with geographic data in another repository [8].

In this study the GLS was designed to mimic the joining of tables in a relational database. The assumptions being both datasets have geographic identifiers to link the two datasets. The result of the joining was a Web-based thematic map distributed as a WMS. Even though the output product was WMS, users had the option of submitting their own

[8] https://www.ogc.org/standards/requests/53

statistical data in CSV format to be integrated with geographic data. In 2010 the GLS was upgraded to Table Joining service (TJS). TJS combine GLS and GDAS rather treating them as separate standards.

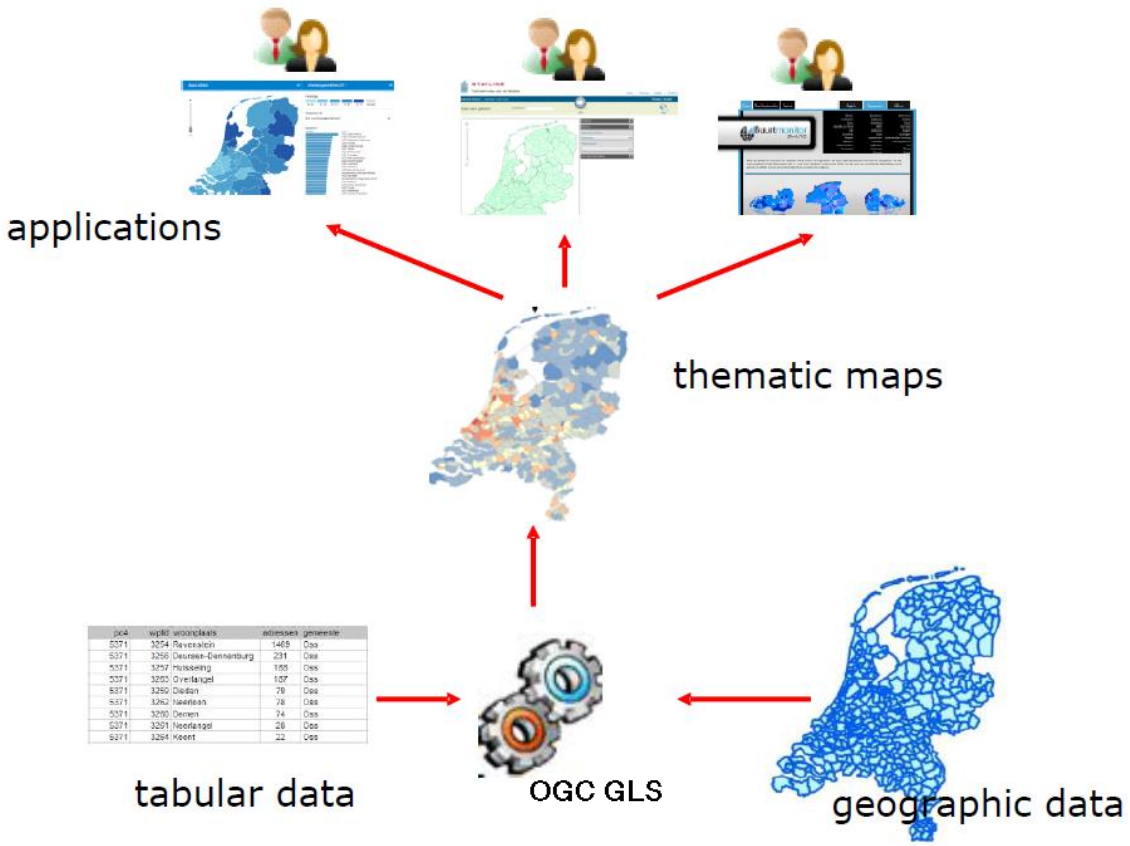

Figure 2. Concept of Geolinking Service (GLS) (Grothe and Brentjens, 2013).

2.1.3 Table Joining Service

The TJS has been around for a little over a decade; however, not much research has been done to demonstrate its potential to join geospatial data and statistical data in a distributed environment. TJS describes a way to define and exchange data that contains information about geographic objects [9]. TJS ingests attribute data which refers to geographic features also known as geolinked data and joins it to a geospatial framework so that it can be mapped as Web based thematic map or used for further geoprocessing.

Attribute data in the context of TJS refers to data about a certain geographic space, but this data is not directly embedded with geographic coordinates or geometry. The attributes are encoded in a format defined by OGC for TJS

[9] https://www.ogc.org/standards/tjs

called Geolinked Data Assess service (GDAS), which is an XML format. The attribute data contains a field with geographic identifiers to indicate the geographic feature to which it applies to.

Geographic identifiers can be administrative identifiers e.g. district name, regions or persistent identifiers like postal code and country area codes. The geographic identifiers are also present in as the framework key of the framework data. Through these common geographic identifiers in the two datasets linking can be realised. The output of the linking operation is a new geospatial framework populated by the attribute data.

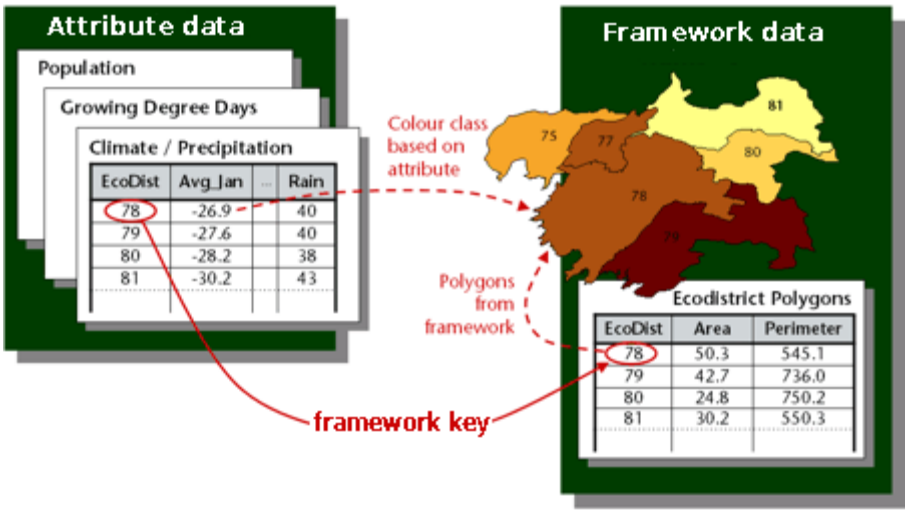

Figure 3 Representation of the framework key present in both attribute data and geospatial framework [10].

Framework data describes data about features positioned on the Earth's surface. These geographic features can be countries, ecological regions, rivers, or census grids. Examples of attribute data that can be related to geographic features include population by country. Framework data may reside locally in a TJS or it can be provided by a Web Service that provides geospatial data in its raw vector format such as the Web Feature Service (WFS). Currently known TJS implementations use the WFS. WFS is an OGC interface standard that allows requesting of raw geographic features over the web [11]. Because in a WFS data are in raw format end users have direct access to geographic information at the feature and feature property level; thus, the data can be edited and used for spatial analysis and geoprocessing. However, WFS does not give administrative rights over the data to clients. Thus, clients without administrative access can only retrieve and modify features virtually but cannot modify the underlying data store containing the original data. This is a major advantage as data can be published once and used many times.

[10] http://geoprocessing.info/tjsdoc/Overview#history
[11] https://www.ogc.org/standards/wfs

2.1.4 Table Joining Service Implementations

In a study by Grothe and Brentjens in 2013 the authors presented merits and possibilities of using TJS. They revealed that very few implementations of the TJS exist. At the time of writing a Google search shows that two TJS software implementations are available (see Table 1). Géoclip is a Web-based platform for geospatial visualization of statistical data, it was developed by a French company called Emc3 [12]. With Géoclip statistical data providers can provide statistical data to be visualized on a web map thanks to TJS.

The second TJS implementation is the open source Geoserver TJS plugin that runs on Geoserver. Geoserver is an open source software to serve maps and spatial data in several formats to clients such as web browsers and desktop GIS applications (Gratier et al., 2015). It offers OGC's Web Services such as WMS and WFS interfaces for building spatial data structures. The TJS plugin implements the joining operation of the WFS with attribute data in GDAS format; the output can be accessed by WMS or WFS.

Table 1. TJS implementations. (Source Grothe and Brentjens, 2013)

Software Products	Client/Server	TJS access/TJS join	Technology	Type of Software
Géoclip	Client server	TJS-access TJS-join	?	Proprietary
Geoserver TJS Extension	server	TJS-access TJS-join	Java	Open source

Bresters et al (2016) conducted an impact analysis of the TJS in an environment comparable to the national infrastructure that the Netherlands uses for the European project INSPIRE. They used the Geoserver TJS plugin. The geospatial framework was provided as Web Feature Service (WFS) and attribute data was first made available in three formats; namely CSV, SDMX and Odata. The attribute data would then be converted to GDAS so that it could be used in TJS. The output of the joining operation of attribute and geospatial framework was accessible WFS or WMS.One of the concerns in this impact analysis is a scenario of when Geoserver changes its versioning. The Geoserver TJS plugin was developed based on the then current Geoserver version. Such tightly coupled software programs have many disadvantages; firstly, when the Geoserver version change the TJS plugin also need to be updated. Secondly the Geoserver TJS plugin is designed to work within Geoserver framework. This does not allow interoperability between the Geoserver TJS plugin and other Web map servers. The other concern was that an additional software to convert the attribute data formats to GDAS was needed. Most attribute datasets are available in

[12] https://www.geoclip.fr/

other formats and there are no tools or specifications of how to convert data to GDAS. The burden of this task is left to attribute data providers.

Another study that examined the TJS concept was by the European and Global Forum for Geography and Statistics (EFGS) and Eurostat in 2019. The study was conducted as a proof of concept for the implementation of the European Union version of GSGF in a project called for the GEOSAT 3 (EFGS and Eurostat, 2019). In this study a TJS implementation was developed based on Map Server. The Geoserver TJS plugin was not of interest because using the Geoserver TJS plugin would require that the geospatial framework be hosted in Geoserver. Geoserver restricts the total count of features and number of requests that can be made to the server. For this study the experimental spatial data used were the 1km by 1km census grids covering GEOSAT 3 countries and Belgium that could be accessed via WFS.

Since the study was done for EU regions the statistical data used followed a format already used for INSIRE datasets called SDMX. EFGS and Eurostat developed the TJS in a manner that it directly ingests attribute data in SDMX format rather than converting it to GDAS first. The output from this research could be delivered by WFS or WMS.

The study demonstrated that TJS offers a solution for joining geospatial data with statistical data in a distributed environment. By using SDMX attribute data the complexity of creating an application for converting SDMX to GDAS was avoided.

One of the major shortcomings faced in this research was transferring and processing geographic features from the WFS. Due to the high volume of features that needed to be accessed, WFS proved to be inefficient as a source for geospatial data as it slowed down the whole system.

Currently to efficiently transport geographic features wrapped in a WFS is to filter geographic feature requested to a limited size so a client is not receiving the whole dataset. In a way this did not take away the concerns that comes with using Geoserver. Another disadvantage of WFS is it only caches request queries for later use and not the geographic features. Caching geographic features would be an advantage in that clients will not need to continually access the server but the cached geographic features. This kind of system is possible when using vector tilling technology.

Vector tiles have been on the geospatial scene for some time now; however, the potential of using them for geoprocessing or spatial analysis is less understood. Most research that focuses on vector tile emphasises on transmission efficiency (Antoniou et al., 2009), optimising vector tile caching (Shang, 2015) and optimising rendering and styling (Fujimura et al., 2019; Netek et al., 2020).

Considering that vector tiles have many advantages over other means of transporting, rendering and storage of geographic features on the web, they hold more promise for future web cartography (Warf, 2018). The following section gives an overview of the vector tilling technology.

2.2 Vector tiles

Vector tiles and raster tiles in WMTS operate the same way. The only difference is instead of breaking down an map image in the case of raster tiles, vector tiles are a product of breaking down vector data (Netek et al., 2020). Like raster tiles, vector tiles also use the TMS and the concept of tile pyramids (Ingensand, 2019).

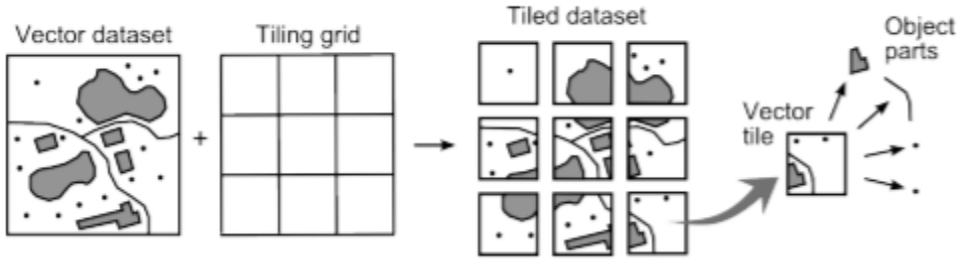

Figure 4 . The procedure for generating vector tiles according to Gaffuri (2012).

The procedure of generating vector tiles involves dividing an original vector dataset into a tiling grid and each corresponding data is displayed on a tile (Netek et al., 2020). After the packaging of the vectors dataset into tiles they now can be transmitted web (Ingensand et al., 2016). Vector tiles are smaller and therefore they download and display more quickly on the client (Kyriakidis et al., 2019, p. 65). Vector tiles are usually named after a specific storage scheme. A storage scheme has parameters such as supported coordinate reference system, size of tiles, number of zoom levels, etc. The web Mercator projection (EPSG: 3857) is the commonly used coordinate reference system (CRS) for web maps; under the Mercator projection a world map is presented as a square with the polar regions cut off. It is this square that represented the zero-zoom level tile in a vector tilling scheme. Subsequent zoom levels are created by dividing the prior tile into four new tiles; as shown on Figure 5 this concept is also known as tile pyramid. If a feature on a tile boundary overlaps into the adjacent tile, the feature is divided, and each section is displayed on the corresponding tile.

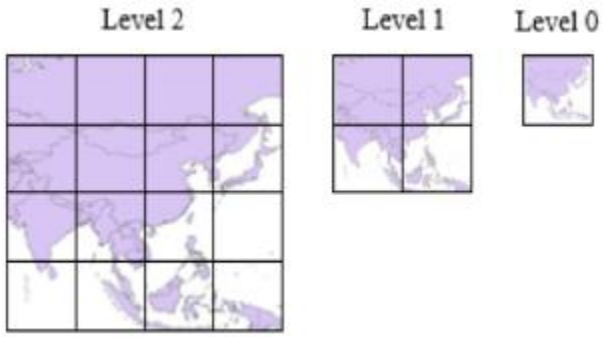

Figure 5. Quad tree structure of a tile pyramid. Source (CSSE, 2014, p. 303).

Vector tiles are requested from a server using an HTTP request and the format for extraction follows a zoom/x/y.format (Sambells et al., 2007, p. 252). For example, to request a tile at zoom level 2 in column 1 and line 1 the format extraction would be 2/1/1.geojson. Vector tiles mainly support three formats namely TopoJSON, GeoJSON and Mapbox Vector tile (mvt) (Shang, 2015).

Vector tiles formats maintains raw vector geometry; therefore, geographic features can be manipulated. Vector tiles are stored on the server side as vector objects and as a result the server does not concern itself with rendering the vector tiles rather the client is tasked with the rendering(Antoniou et al., 2009). The other interesting feature of vector tiles is that they can be cached. Tile caching is a technique that allows the map server to generate vector tiles and store them for future use. As a result, map servers can pass the vector tiles to the client immediately without querying from the server. Caches reduce demand on the GIS and frequent access to database servers (Zouhar and Senner, 2020). Additionally, caches can be created for the client so that the vector tiles already downloaded from a server can be reused without downloading again enabling, an optimised transmission over a network. The tile caching is suitable for map data that do not change frequently such as administrative boundaries.

3. Methods

3.1 Case study scenario

Of late, interest has been sparked in implementations for better integration of spatial and statistical data. For example, this a has led to establishment of the Global Statistical Geospatial framework for the homogeneousness production and integration approaches for geospatial and statistical data. The incentive being anticipation of the statistical results collected for SDG indicators and how to geovisualise them.

A case study scenario that would require this approach follows. The United Nations Statistical Division (UNSD) collects statistical data of Sustainable Development Goals (SDG) indicators for member states. For example, hypothetically Gross Domestic Product (GDP) per capita can be used as indicator for economic growth in line with SDG 8. The statistical data of GDP for member states is collected, stored and published on the web as web resources in CSV format [13].

At the same time the published CSV data may not always contain other supporting additional attributes .For example instead of utilising just the GDP data one may want to standardize the GDP values using the country population or a country's surface area. This supporting attribute data can be queried from Webs of data on the Semantic Web. Just as there is need for a geographic identifier for merging statistical and geospatial data, there is also a need for these identifiers to merge and query attribute datasets that are stored on different RDF stores on the Semantic Web. The dataset can be queried using SPARQL; the results of the query can be converted to a CSV format.

Statistical data is often collected and aggregated according to administrative boundaries, for example at a national level. Thus, each administrative boundary can have multiple attributes like name of the country, its population and various other statistics. Statistical data collected at a national scale, especially for the UN member states, often contain a field with a three-digit area code developed and maintained by UNSD. The three-digit area code is called UN M49 or the Standard Country or Area Codes for Statistical Use (Series M, No. 49) [14]. It is also the same as the ISO 3166-1 numeric standard published by the International Organization for Standardization (ISO). This three-digit code is unique for each member state and is useful as a form of geographic identifier to link various datasets from different sources including geographic features in a GIS environment.

The working assumption is that spatial data is published as vector tile format and can be assessed by other programs on the internet as a map web service. The goal is to populate the vector tiles with statistical data so it can be transported and displayed on a web client as a web thematic map.

[13] http://data.un.org/
[14] https://unstats.un.org/unsd/methodology/m49/

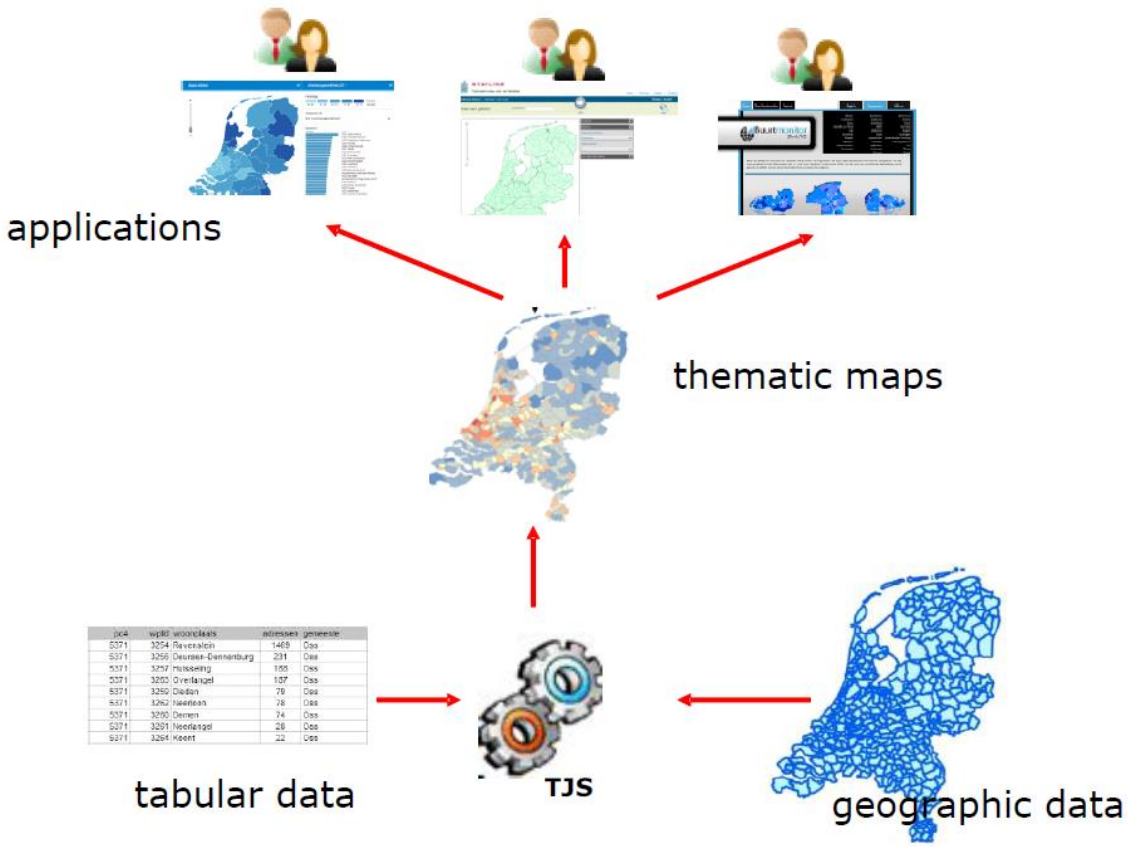

Figure 6. Table Joining Service concept for automated service-oriented data joining (Grothe and Brentjens, 2013).

Another assumption to note is the software architectural style for the web services is RESTful. RESTful web services are web services that adhere to a Representational state transfer (REST). This means web services allow other programs on the internet to access and manipulate the textual representation of web resources (Masse, 2011, p. 6). The textual representation of web resources includes the GeoJSON (a vector tile format) and CSV data formats. In a RESTful system web resources can be requested using a Uniform Resource Identifier (URI) by means of Hypertext Transfer Protocol (HTTP). A URI is a short string that explicitly identifies a resource on the web.

HTTP is the primary means of communication on the web. HTTP has methods that indicate which direction data is moving and what should happen to it. These methods include GET, POST, PUT and DELETE. For example, GET method is used to request data from a specified resource. POST method is used to send data to server to create or update a resource (Ashton Acton, 2013).

RESTful web services also use stateless protocols, meaning a server does not need to retain information about a request connection between client and server. A connection is only valid whilst a transaction is running. This is beneficial

because software components can be managed and updated without affecting the entire system. In short SOA is used with REST to ensure rapid performance and reliability.

3.2 Software architecture

To provide a proof of concept for TJS concept, a prototype with a three-tier architecture is developed. A three-tier architecture is a client-server software architecture with software components divided into three layers. Each layer is established according to the software component functionality. These three layers are the presentation layer, the application layer and the data layer. The reason for separating the layer is software components can be developed and maintained as autonomous components on separate platforms (Tiwari and Jain, 2014). The advantage of using this architecture is it would be possible to switch or alter technologies used at each layer without affecting the whole system (Krewinkel et al., 2015). The following sections describe the functions of each layer and the proposed software components.

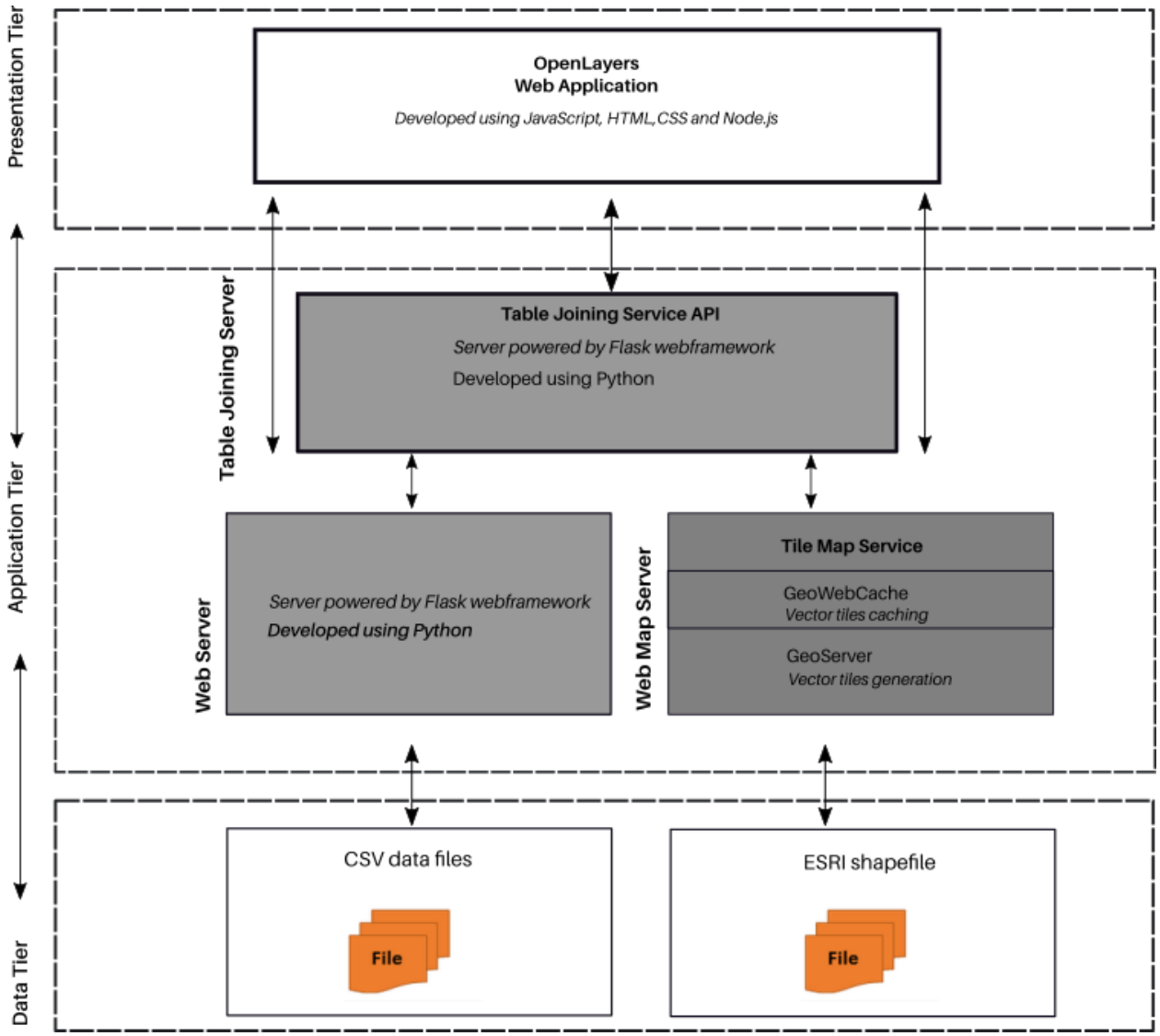

Figure 7. Prototype software architecture.

3.2.1 Presentation tier

The presentation layer, also referred to as the client layer, provides the user interface that consumers interact with. This layer is the front end of the entire system and users can access the system via a Web browser. For this prototype users will be able to access the results from the TJS joining operation and display them in Web Map Application. This layer is be supported by technologies which include JavaScript, HTML, CSS and Node.js. The presentation layer is if the topmost layer on a hierachy of and application and its communication with other layers via API requests.

3.2.2 Application tier

The application layer contains the business logic applications a system's core functions capabilities. For this prototype on the application layer are three autonomous server components, namely a Table Joining Server, a Web Server and a Web Map Server. The servers contain Web services and Web resources that can be requested over HTTP protocol. The main reason for separating the software components is the assumption that each server is provided by the service provider independently. The functions of each server component are described below.

Table Joining Server is the physical instantiation of the Table Joining Service. It provides the Application Programming Interface for joining vector tiles from the Web Map Server and attribute data from the Web Server. The results of the joining operation are vector tiles populated with the attribute data and published as a Tile Map Service. The results can be accessed by Web Map Application in the presentation layer. The Table Joining server and the Table Joining Service are built using Flask, a micro Web framework.

Web Server is dedicated to hosting the attribute data in CSV format. It also containing interface that enables the server to access SPARQL Endpoint services and access RDF data stores. The queried results are converted into CSV data formats and stored inside the Web Server. Requests can be made to access CSV data using the HTTP protocol. The server is built using Flask a micro Web framework.

The Web map server is dedicated to creating and publishing geospatial Web services. Geospatial data from the data layer can be assessed by other layers as Web services. Geoserver is used as the web map server. Vector data from the data layer is used to generate vector tiles using a Geoserver Vector Tiles extension. A software component integrated in Geoserver called GeoWebCache is used to cache vector tiles and publish vector tiles as a Tile Map Service.

3.2.3 Data tier

The data layer represents the container of input data that can be used by the system. For this prototype the data is uploaded to servers in the application layer from data files located on the computer hard disk. Geospatial data is available in ESRI shapefile format and attribute data as CSV data formats.

3.3 Software technologies

All software and programming language used develop the software prototype to achieve the objectives of the thesis are open source and can be downloaded for free. The motivation for using open source technologies is to guarantee that anyone can read, modify, and build on this concept to improve the quality of the software so that it can be redistributed at no cost.

The prototype is built on a Windows 10 Operating System and Google Chrome as the web browsers all web applications are executed. Software, technologies and programming language used to develop the prototype are described in the tables below.

Table 2. Operating system and web browser.

COMPONENT	TECHNOLOGIES	PROGRAMMING LANGUAGE LIBRARY	FUNCTION
OPERATING SYSTEM	Windows 10 64-bit		Software running on computer hardware.
WEB BROWSER	Google Chrome		Retrieving and displaying contents from the Web server.

Table 3. Software and technologies for the client tier component.

COMPONENT	TECHNOLOGIES	PROGRAMMING LANGUAGE LIBRARY	FUNCTION
WEB MAP APPLICATION	HTML		A standard mark-up language for displaying contents in a web browser. In this prototype it carries content of the Web map applications that should be displayed on the browser.
	CSS		A language to describe how elements in the HTML document should be displayed including styling and a structure of the web page.
	JavaScript (JS)	OpenLayers (OL) JavaScript Library v6.4.3	A programming language that can create functions that control behaviour of a Web document. The main JS library used in this thesis is the OL. It is used to display the vector tiles from the TJS.
	Node.js		Acts as backend JS runtime environment. For this prototype it executes OL JS script to produce vector tile maps in real time.

Table 4. Software and technologies for the application tier component.

COMPONENT	TECHNOLOGIES	PROGRAMMING LANGUAGE LIBRARY	FUNCTION
WEB SERVER	Python 3.8.5 and a Web framework	Flask	Flask is web framework building Web servers. It is used to build a mock Web server for storing CSV data files. It also manages HTTP requests for the Web resources in the Web Server. Flask is purely built using Python. Python is a high-level object oriented programming language for building software.
	Python 3.8.5, SPARQL Endpoint SPARQL	SPARQLWrapper	A python module for querying data from SPARQL endpoint using SPARQL. SPARQL endpoint is a web exchange service that provides access to RDF data stores. SPARQL is a language for querying RDF data stores.
		Pandas	Pandas is a python module that would be used to convert SPARQL query results to CSV data format.
WEB MAP SERVER	GeoServer 2.15	Geoserver Vector Tile Extension	GeoServer Java web application for implementing OGC protocols and other geospatial services. It is to create vector tiles and publish them as a TMS.
	GeoWebCache 1.15.0 (GWC)		A Java web application that will be used to cache vector tiles. GWC integrated in Geoserver is used.
	Tile Map Service		Open standard for publishing and distributing tiled maps.
TABLE JOINING SERVER	Table Joining Service: Python 3.8.5 and a Web framework	Flask Framework	Flask is web framework building Web servers and Web services. It is used to build the Table Joining Service API for

			joining cached vector tiles with attribute data. It is also used build the Table Joining Server where the TJS API can access from.
		Pandas	Pandas is a python module that would be used to manage and process CSV data to be used the TJS data operation.
		Geopandas	GeoPandas is a python module that would be used to manage and process cached vector tiles in GeoJSON format to be used the TJS data operation.

Table 5. Software and technologies for the data tier component.

COMPONENT	TECHNOLOGIES	PROGRAMMING LANGUAGE LIBRARY	FUNCTION
DATA FILES	CSV		CSV is simple data format for data in tabular format.
	ESRI Shapefile and QGIS 3.4.6		ESRI Shapefile is a common data format for vector data. It can be processed and displayed in a QGIS. QGIS is a GIS desktop application.

4. Prototype Implementation

To examine the feasibility of using vector tile cache as a framework data for TJS a prototype application is developed based on the three-tier architecture described previously. In this section the process of developing the prototype is described. The input experimental data used and the output experimental results from the data joining operation using the developed TJS API are also described. Furthermore, a description of the development of the Web map application that would be used to display the results of the TJS data join is described.

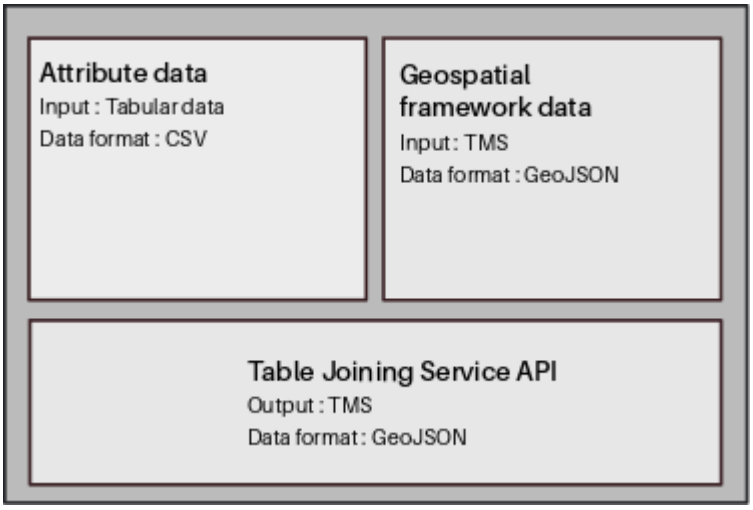

Figure 8. Input and Output data formats for Table Joining Service.

4.1 Experimental data

The following sections give a description of the experimental data including preprocessing procedures for each dataset.

4.1.1 Attribute data

Attribute data used is retrieved from two sources: Wiki data SPARQL endpoint [15] and a CSV data file available from UNDS website [16]. The attribute data from the two sources can be used independently or merged to get enriched attribute data. Any joining procedures for the attribute dataset is done using the python module called Pandas. The results are stored in a mock Web server developed using the Flask.

Wiki data SPARQL endpoint

SPARQL endpoint provides access to RDF data stores on the Web. SPARQL query language is used to query RDF data. To automate the process of running a SPARQL query and converting the results into a CSV data file the python

[15] https://query.wikidata.org/
[16] http://data.un.org/

module called SPARQLWrapper is used[17]. SPARQLWrapper provides an interface for SPARQL endpoint services to be queried in a Python environment. The results from querying a Wiki data SPARQL endpoint service were then converted to CSV data formats. The following SPARQL query retrieve data from a Wiki data SPARQL endpoint.

```
1. SELECT distinct? countryLabel? UN_A3? ISO_A2? Population
2. WHERE
3. {
4.   ? country wdt: P299? UN_A3.
5.   ? country wdt: P297? ISO_A2.
6.   ? country wdt: P1082? Population.
7.   SERVICE     wikibase:label     {bd:serviceParam     wikibase:language
   "[AUTO_LANGUAGE], de”}
8. } order by? ISO_A2
```

The results of the query are in a tabular format. Shown here are the first five rows of country records queried out of 251 rows.

Ländername	UN_A3	ISO_A2	Population
Andorra	020	AD	76177
Vereinigte Arabische Emirate	784	AE	9400145
Afghanistan	004	AF	34940837
Antigua und Barbuda	028	AG	102012
Anguilla	660	AI	16086
Albanien	008	AL	3020209

Figure 9. Results from SPARQL query.

United Nations Statistical Division (UNSD) website

UNDS website provided statistics on national Gross Domestic Product (GDP) for UN member states. The data is available in Comma Separated Value (CSV) format. CSV data format for attribute data provides a table like structure and uses commas or other characters to separate column values from one another. The original dataset contained gross domestic product and gross domestic product per capita values for UN regions and UN member for years between 1985 and 2017. For simplicity the data was reduced to only GDP for the year 2005. To achieve this the CSV file was imported into a Postgres SQL database and using a SQL query statement only 3 columns retrieved.

```
1. SELECT GDP, Country_name, UN_3
2. FROM public.gdp_countries
3. where year=2005
```

[17] https://github.com/RDFLib/sparqlwrapper

The results of the query are in a tabular format. Shown here are the first five rows of country records queried out of 222 rows. The CSV data was then uploaded to a mock server developed using micro web framework written in python called Flask.

UN_A3	Country_name	GDP(2005)
4	Afghanistan	17140
8	Albania	19895
12	Algeria	248534
20	Andorra	48395
24	Angola	94874
660	Anguilla	18627

Figure 10. Results from SQL query.

Joined Attributes

The two attribute datasets can be merged based on the common identifiers i.e. ISO 3166-1 numeric standard codes UN_A3.

UN_A3	Country_name	GDP(2005)	ISO_A2	Ländername	Population
4	Afghanistan	17140	AF	Afghanistan	34940837
8	Albania	19895	AL	Albanien	3020209
12	Algeria	248534	DZ	Algerien	41318142
20	Andorra	48395	AD	Andorra	76177
24	Angola	94874	AO	Angola	29784193

Figure 11. Results from the joined attribute data.

4.1.2 Vector data for the TJS geospatial framework

The vector dataset used to create vector tiles came from Natural Earth in the ESRI shapefile format [18].Natural Earth is a public domain providing geospatial data available at different scales. For this study, the 1:10million vector datasets of countries were used. The vector dataset is embedded with attributes for each geographical feature representing a country. These attributes include the Series M, No. 49 standard area codes that are also used by UNSD to collect statistical data. This attribute will be used as the geographic identifier. Figure 12 shows a portion of the attribute table of the vector dataset displayed in a QGIS application.

[18] http://www.naturalearthdata.com

ISO_N3	UN_A3	REGION_UN	SUBREGION	REGION_WB	NAME_DE	NAME_EN
004	004	Asia	Southern Asia	South Asia	Afghanistan	Afghanistan
008	008	Europe	Southern Europe	Europe & Centr...	Albanien	Albania
012	012	Africa	Northern Africa	Middle East & ...	Algerien	Algeria
024	024	Africa	Middle Africa	Sub-Saharan Af...	Angola	Angola
031	031	Asia	Western Asia	Europe & Centr...	Aserbaidschan	Azerbaijan
032	032	Americas	South America	Latin America ...	Argentinien	Argentina
036	036	Oceania	Australia and N...	East Asia & Paci...	Australien	Australia
040	040	Europe	Western Europe	Europe & Centr...	Österreich	Austria
044	044	Americas	Caribbean	Latin America ...	Bahamas	The Bahamas
050	050	Asia	Southern Asia	South Asia	Bangladesch	Bangladesh
051	051	Asia	Western Asia	Europe & Centr...	Armenien	Armenia
056	056	Europe	Western Europe	Europe & Centr...	Belgien	Belgium
064	064	Asia	Southern Asia	South Asia	Bhutan	Bhutan
068	068	Americas	South America	Latin America ...	Bolivien	Bolivia

Figure 12. Attributes of the Natural Earth vector dataset displayed in QGIS application with the column of the Series M, No. 49 UN area codes highlighted.

Pre-processing of the vector data was done in a QGIS. The vector dataset contained some outdated M49 area codes for some countries which needed to be updated. This vector data would then be uploaded to Geoserver to generate vector tiles.

4.2 Geospatial framework data

The geospatial framework for the TJS can be accessed via a Tile Map Service (TMS). TMS is a Web service that supports publication of tiled maps in various formats. For the prototype, a tiled GeoJSON is published via TMS. GeoJSON is a data format for vector data and it is also one of the common formats for vector tiles. The following sections describe processes, methods for generating and caching vector tiles using GeoServer and GeoWebCache.

4.2.1 Generating vector tiles with GeoServer

GeoServer Vector Tile Extension was used to generate vector tiles. Vector Tile Extension is a module that adds a functionally to Geoserver for vector tile generation. It is installed as an add-on to the base Geoserver installation. [19]. Geoserver is an open source software to serve maps and spatial data in several formats to clients such as web browsers and desktop GIS applications (Gratier et al., 2015).

GeoServer generates vector tiles in three major formats which are MapBox Vector (MVT), GeoJSON and TopoJSON [20]. MVT format is a commonly used format for encoding vector tiles and is supported by most current web map

[19] https://docs.geoserver.org

[20] https://docs.geoserver.org/latest/en/user/extensions/vectortiles/tutorial.html

application (Netek et al., 2020). However, MVT is optimised for rendering and does not specify how the format can be used as a dataset [21]. Therefore, for this study the GeoJSON format is of interest because of its JavaScript Object Notation (JSON) structure.

GeoJSON data format is already supported by programs for feature manipulation and feature analysis on the web (Balog and Houtmeyers, 2017). GeoJSON is a geospatial data interchange format based on JSON (Gillies et al., 2016). Geographic features are presented as a combination of JSON objects along with their properties and spatial extents. GeoJSON geometry types are: Point, Linestring, Polygon, MultiPoint, MultiLineString, MultiPolygon and GeometryCollection [22]. A geographic feature in GeoJSON is represented as a Geometry object and a feature collection is a list of features (see Figure 13).


```
"features":[
  {
    "id":"0",
    "type":"Feature",
    "properties":{
      "UN_A3":242,
      "Country_name":"Fiji",
      "GDP(2005)":"9641"
    },
    "geometry":{
      "type":"MultiPolygon",
      "coordinates":[
        [
          [
            [
              20037508.34,
              -1812498.41
            ],
            [
              20037508.34,
              -1869110.46
            ],
```

```
{
  "type":"FeatureCollection",
  "features":[
    {
      "id":"0",
      "type":"Feature",
      "properties":{ },
      "geometry":{ }
    },
    {
      "id":"1",
      "type":"Feature",
      "properties":{ },
      "geometry":{ }
    },
    {
      "id":"2",
      "type":"Feature",
      "properties":{ },
      "geometry":{ }
    },
```


Figure 13 GeoJSON object (left) and FeatureCollection (right)

4.2.2 Caching vector tiles with GeoWebCache

Any application that requests for vector tiles from Geoserver are redirected to the GWC endpoint where vector tiles are cached. GWC cache vector tiles on disk. The tile caching is activated by enabling the GWC Tile Map Service endpoint in the Geoserver application:

```
http://localhost:8080/geoserver/gwc/service/tms/1.0.0?
```

[21] https://docs.mapbox.com/vector-tiles/specification/
[22] https://geojson.org/geojson-spec

The URL the above lists all dataset layers cached within GWC as vector tiles. To get a specific vector layer the request must specify the layer name, coordinate reference system and the format of the tiled map.

```
http://localhost:8080/geoserver/gwc/tms/1.0.0/layername@grisetId@formatExt
ension/z/x/y.format

z is zoom level

x and y define a given tile coordinates

gridsetId refers to the coordinate reference system

format refers to format of the vector tiles
```

If the requested vector tiles are not found in the cache storage folder, contact will be made to Geoserver. The GWC is populated dynamically as responses are transferred back to a web client. GWC also allows for pre-seeding of vector tile cache. Future requests are made to the cache storage that is already populated with pre-generated vector tile.

4.3 TJS Application Programming Interface implementation

To facilitate the joining of the geospatial framework and the attribute data a RESTful API was created based on the TJS JoinData operation. The API pull GeoJSON vector tiles and the CSV attribute data from GWC and the attribute data server. Once the data is pulled a function provided by the API processes and merges the two datasets. After the data has been merged and converted to GeoJSON format the OpenLayers web application can make requests to the TJS server and render the results as a vector tiled map.

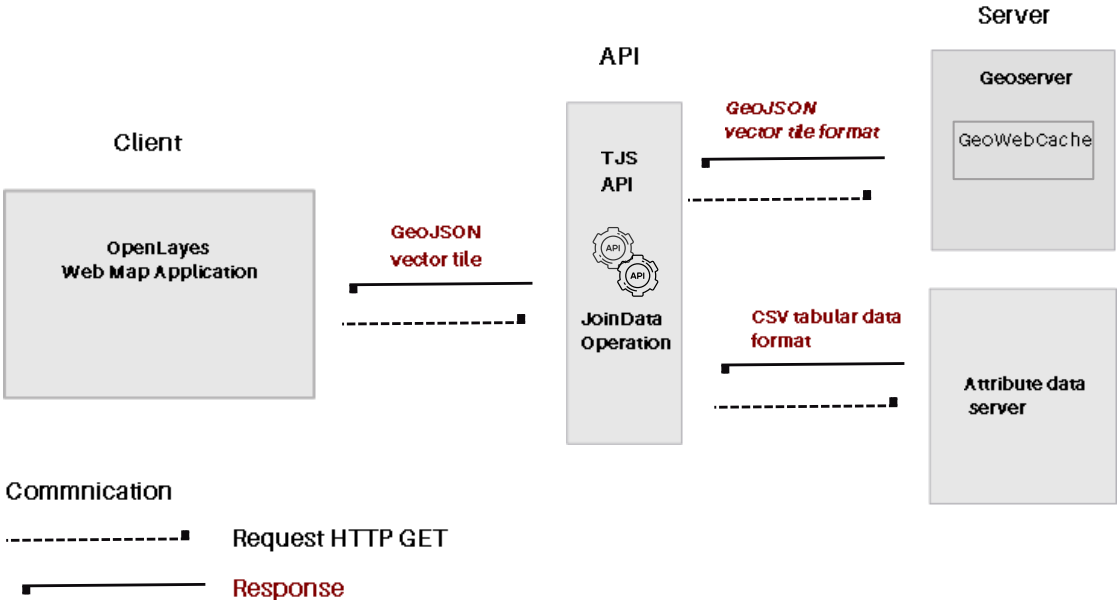

Figure 14 . TJS JoinData operation procedure steps RESTful system.

The following sections give a description of how the API was built and the function that automates the joining of the two datasets. First a description of the logics of the Python function for the joining operation is given. A Python function is a block of code written in Python programming language that is executed when invoked. Secondly a description of how the TJS API for data joining was developed. The API calls the Python function via a URL and receives certain parameters for it to return joined data.

4.3.1 Python function for joining data

The function is for joining the geospatial data and the attribute data is built using three Python modules. These modules are Request, GeoPandas and Pandas. Note the Request modules can be used integrated in the Flask module or can be used independently.

A module is a code library which contains a set of functions that can be used in an application. The function is built so that it accepts at least four parameters which are: the URL to get the vector tiles, URL to get attribute data in CSV format, the name of the geographic identifier, also called the framework key and at least one attribute from the CSV table. These parameters are listed in the parenthesis of the functions; they are values required by the function to execute. Figure 15 illustrates how the function works to return results of the joining.

Request module allows users to send HTTP request using Python. The request returns a response object with all the response data[23]. The vector tiles URL parameter is used to make requests for the GeoJSON vector tiles. The attribute URL requests the CSV attribute data. Pandas to read CSV data and Geopandas reads GeoJSON data.

[23] https://pypi.org/project/requests/2.7.0/

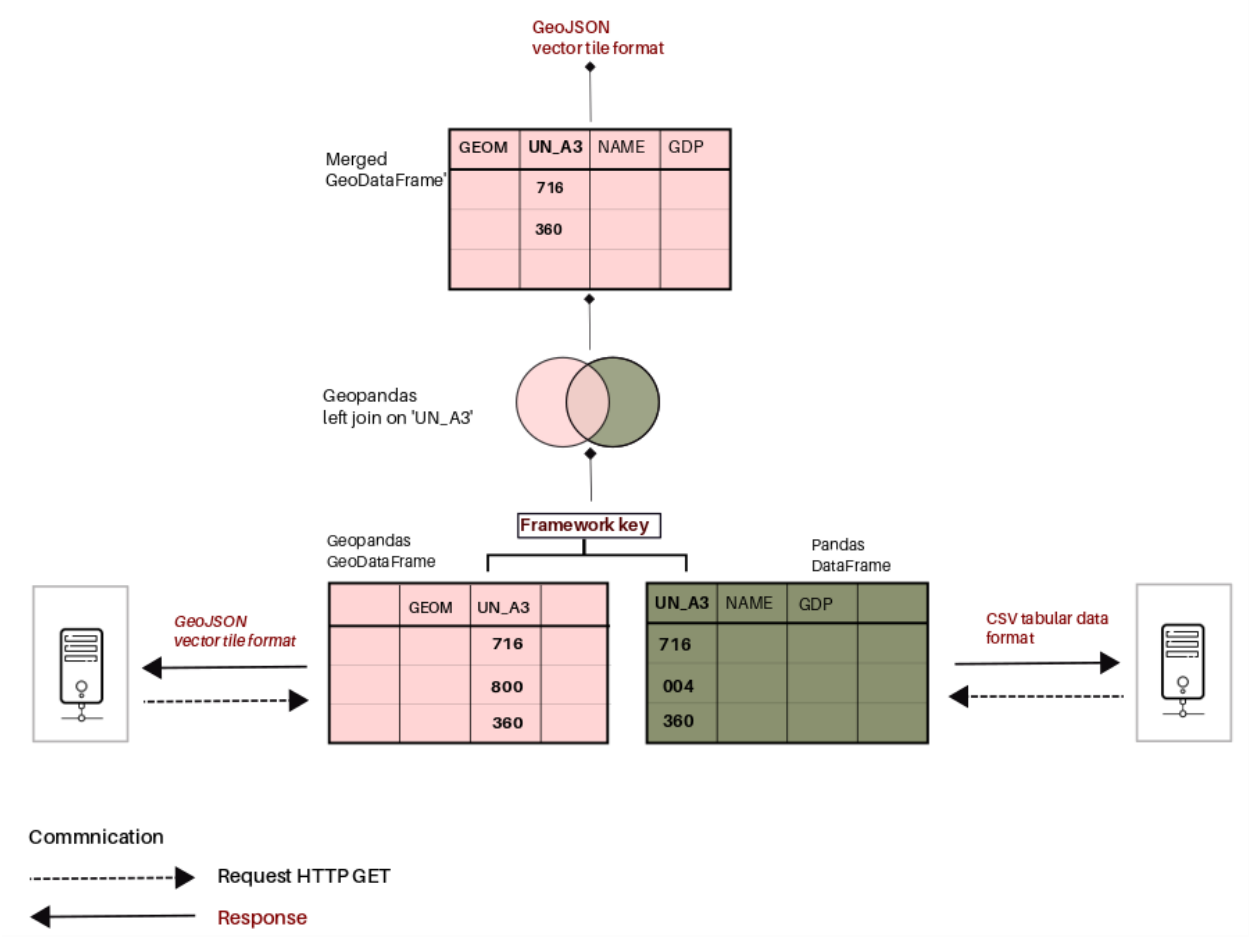

Figure 15. Procedure of joining operation

Pandas is an open source Python library that allow data analysis and manipulation. It has methods and functions that allow reading, processing, and writing data from CSV format [24]. The Pandas library converts CSV data into a data frame.

Geopandas is an open source Python library that works with geospatial data. GeoPandas supports processing and analysis of geospatial data types including the GeoJSON format. It encompasses datatypes used by Pandas thus allow spatial operations on geometric types [25]. The GeoPandas is tasked with converting the GeoJSON to a data frame and the merging the Pandas data frame and the Geopandas data frame into one.

GeoPandas allow attribute join using its merge method. For the final data frame to retain the geometry field, it is recommended that join be on the data frame from the GeoJSON. After the merge the data frame is converted back to GeoJSON vector tile, where the vector tiles can be accessed from URL that is mapped onto the joining function. Flask

[24] https://pandas.pydata.org/
[25] https://geopandas.org/

is a web micro framework for building web APIs. It has functions that map a URL path to a Python function running on a Flask web server.

The joining of the data frame from GeoJSON and data frame from CSV is done based on common geographic identifier. Vector tile in Geojson format and attribute data in CSV format are requested from their respective servers. GeoJSON vector tile is converted into a geodata frame inside the Python function using GeoPandas and the CSV is converted into a data frame using Pandas. The difference between a regular Pandas data frame and geodata frame from Geopandas is the geodata frame contains a field with feature type and geographic coordinates of that feature (see Figure 16 and Figure 17).

geometry	UN_A3	NAME_EN
POLYGON ((20037508.34 -1812498.41 ….	242	Fiji
POLYGON ((3774143.87 -105758.36 ….	834	Tanzania
POLYGON ((-964649.02 3205725.61	732	Western Sahara
POLYGON ((-13674486.25 6274861.39	124	Canada
POLYGON ((-13674486.25 6274861.39	840	United States of America

Figure 16. GeoPandas data frame from GeoJSON data

UN_A3	Country_name	GDP (2005)
4	Afghanistan	17140
8	Albania	19895
12	Algeria	248534
20	Andorra	48395
24	Angola	94874

Figure 17. Pandas data frame from the CSV data.

Figure 18 shows the results of the merged attribute in the new geodata frame. All rows with geographic features with a geographic identifier that do not match the geographic identifier in the data frame are discarded. Geopandas would then convert the data frame into a GeoJSON format that can be accessed by any web client. Figure 19 shows the before joining and after and after GeoJSON for a geometric feature.

geometry	UN_A3	Country_name	GDP(2005)
POLYGON ((20037508.34 -1812498.41 ….	242	Fiji	17140
POLYGON ((3774143.87 -105758.36 ….	834	Tanzania	19895
POLYGON ((-964649.02 3205725.61	732	Western Sahara	248534
POLYGON ((-13674486.25 6274861.39	124	Canada	48395
POLYGON ((-13674486.25 6274861.39	840	United States of America	94874
POLYGON ((20037508.34 -1812498.41 ….	398	Kazakhstan	18627

Figure 18. Resultant merged geodata frame.

```
{
   "type":"FeatureCollection",
   "features": [
{
         "id":"2",
         "type":"Feature",
         "properties": {
            "UN_A3":732,
            "Country_name":"Western Sahara",
              "GDP(2005)":"2178617"
          },
          "geometry":{
             "type":"Polygon",
             "coordinates":[
                [
                   [
                      -964649.02,
                      3205725.61
                   ],
                   [
                      -967065.11,
                      2984356.69
                   ],
                   [
                      -1332429.62,
                      2990828.74
                   ],
                   [
                      -1439261.04,
                      2430921.27
                   ],
                   [
                      -1899491.58,
                      2391849.03
                   ],
                   [
                      -1642068.75,
                      2451670.88
                   ],
                   [
                      -1268213.41,
                      3108914.65
                   ],
                   [
                      -964649.02,
                      3205725.61
                   ]
                ]
             ]
          }
       },……————————————————————,]}
```


Figure 19. Illustration of feature properties for a geographic feature after the joining operation.

A web map application can request for the GeoJSON vector tiles now populated with attributes from the CSV data frame using the URL. The following sections describes the designing of the URL for the TJS API where web clients can request the joined data from.

4.3.2 Developing the TJS API

The prototype API was developed using Flask web. Flask web is a micro web framework written in Python[26]. It supports the development of web applications, webservices, web APIs and web resources as well as managing HTTP requests. In this prototype development HTTP GET requests are of concern as they correspond to reading data from servers.

The design principle of this prototype greatly revolves around a well formatted request. For example, a Flask server at:

```
http://127.0.0.1:5000/
```

Flask maps HTTP request to a Python functions (TJS JoinData operation) in a process called routing. The syntax below:

```
@app.route(/tjs/api', methods=['GET'])
```

informs Flask that a function should be mapped to **/tjs/api**. The syntax (**methods=['GET']**) reveals the kind of HTTP request allowed in order to access the function. The URL to access the TJS api would be written as follows:

```
http://127.0.0.1:5000/tjs/api?
```

TJS requires a HTTP GET response for a specific vector tile dataset and a specific attribute dataset from severs. A general resolution is to filter results of a request is to add a query string into the URL. The data passed through the URL after the question mark symbol (?) are called query parameters with a key-value- pair encoding (KVP). The URL below shows the HTTP GET request of the JoinData operation request using the KVP encoding implemented by the Flask server.

```
http://127.0.0.1:5000/tjs/api?

FrameworkURI=http://localhost:8080/geoserver/gwc/service/tms/1.0.0/countri
es%3Ane_110m_admin_0_countries@EPSG%3A3857@geojson/{z}/{x}/{y}. geojson&

GetDataURL=http://127.0.0.1:8000/static/sample-csv.csv&

Framework Key=UN_A3&

attribute1=Country_name&

attribute2=GDP (2005)
```

[26] https://flask.palletsprojects.com/en/1.1.x/

4.4 OpenLayers web application.

For displaying the joined results, a simple WebGIS application was built using OpenLayers JavaScript library. The OpenLayers web application was executed on a Node.js server environment. Node.js is open source; it allows creation of web servers, run a JavaScript code outside a web browser. The styling is done on the client side, the products being a choropleth map showing GDP values for the year 2005 over countries. The styling is basically colouring the countries with different colour shade to show the relative differences in GDP per capita.

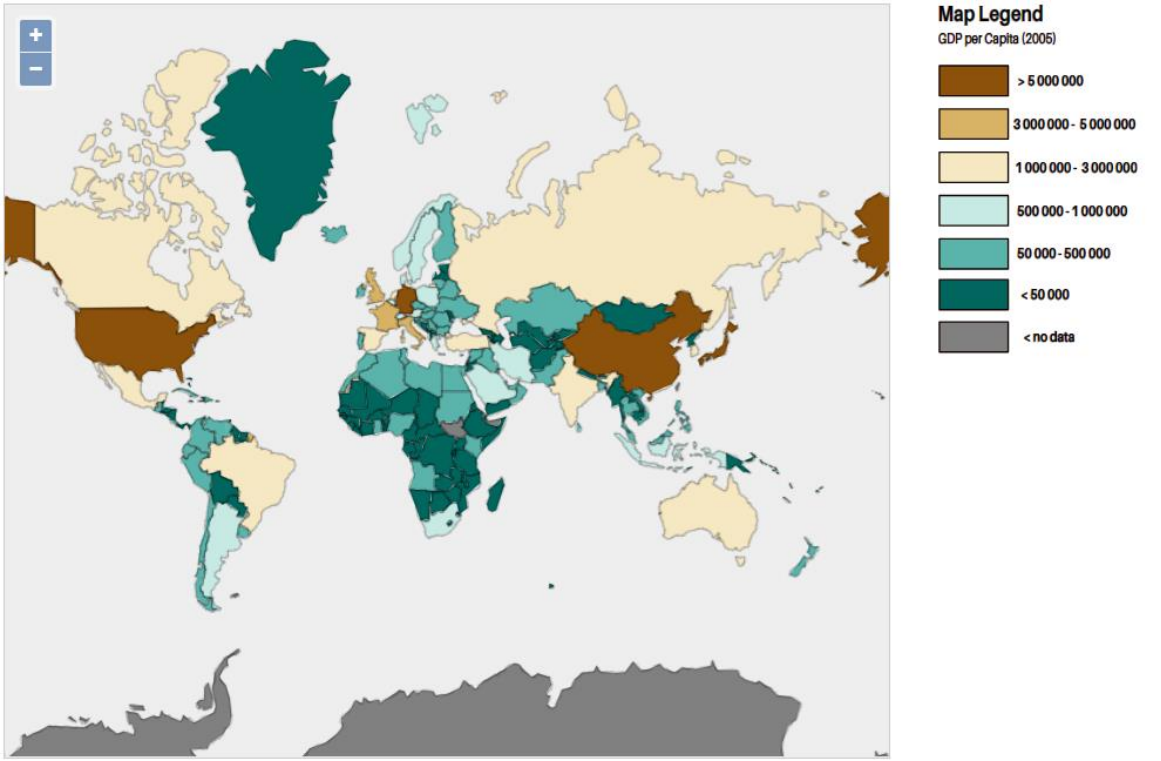

Figure 20. Rendered GeoJSON vector tiles from the resultant TJS joining operation.

5. Discussion

This principal goal of this study is to examine the feasibility of using cached vector tiles as a geographic framework in TJS joining operation. TJS is used as a tool for joining geospatial data with attribute data in a distributed environment. The attribute data is sourced from RDF data stores and converted to CSV data format. A TJS prototype implementation was developed to facilitate joining operation. For a TJS joining operation to be implemented at least four factors are important. These four factors are the geospatial framework data, framework keys, attribute data and the software architecture. In this section each factor is discussed in relation to this study and previous that provided TJS proof of concept.

5.1 Vector tile cache as framework data

The possibility of using cached vector tiles as geospatial framework data is examined. Vector tiling is confirmed as a mechanism for optimal storage, efficient transmission and optimised rendering of geographic vector data from large datasets on the Web ("OGC Vector Tiles Pilot: WFS 3.0 Vector Tiles Extension Engineering Report," 2019). The successful joining of attribute data with vector tile caches serves as a proof of concept. Previous studies by Bresters et al., 2016 and EFGS and Eurostat, 2019 conducted to examine the TJS concept used WFS as the geospatial framework. WFS publish geographic data in several formats such as Geographic Mark-up Language and GeoJSON [27]. EFGS and Eurostat, (2019) observed that WFS with geographic features encoded using GeoJSON format performed better than those encoded in GML. Nevertheless, the general shortcoming experienced whilst using WFS was increased loading, processing and transmission time. This was particularly an issue with the study by EFGS and Eurostat, 2019; they used 1 km^2 by 1 km^2 census grid covering many countries within the EU region. Dealing with such a large dataset is cumbersome and require more computational resources.

This study used vector tiles encoded in GeoJSON format; whether vector tiles are undeniably efficient over the WFS used in current TJS application is yet to be tested. But based on the technologies and mechanisms employed for creating and publishing vector tiles, they may offer solution some solutions to some of the current issues with WFS as geospatial framework data for TJS. The advantages of vector tiles over WFS includes their ability to cached efficiently ,ability to be transmitted speedily and a decreased loading rate which can be attributed to the small size of vector tiles (Shang, 2015). These characteristics of vector tiles that makes them worth considering as a source of geographic framework for TJS.

5.2 Attribute data

The current version of TJS only support attribute data encoded in GDAS format. However based of previous studies there has been inconsistences in attribute data formats used. Hong and Lin, 2005 used the CSV data format and EFGS and Eurostat, 2019 used the SDMX data format. A study to examine the impact analysis of using TJS for Statistics

[27] https://www.ogc.org/standards/wfs

Netherlands had attribute data in three formats namely SDMX, Odata and CSV. The authors developed an API that coverts attribute data coming from the various formats to GDAS (Bresters et al., 2016).

This study used CSV data format. CSV data format is the most popular data format for publishing data on the Web [28] and in Open Government Data (OGD) portals (Mahmud et al., 2020). CSV data can be published as Linked Open Data (LOD) on the Web. LOD can also be accessed from the Web and converted into CSV data. Over the years, the amount of LOD published on the Web has exploded. A majority of LOD is embedded with geographic information. This type of LOD data is called Geolinked data and it is at the core of geospatial artificial intelligence geoAI.

Attribute dataset accessed from Wiki data SPARQL endpoint service was one of the sources of attribute data. A SPARQL endpoint service gives access to LOD stored in RDF datastores. Using SPARQL query language attribute data, that can be linked to geospatial web services through a geographic identifier, was retrieved. It was vital to find data embedded with the ISO 3166-1 numeric identifiers that could be used as geographic identifiers [29]. Persistence identifiers are unique names given to a resource. ISO 3166-1 numeric codes are standardised persistence identifiers that uniquely label countries [30].

One short coming discovered when accessing attribute data from SPARQL endpoints was limited efficiency in accessing data. RDF data stores also experience high access rates like web map servers. When users try to access large datasets, it becomes even more cumbersome for data servers (Akhtar et al., 2020). Vector tiles support transmission of vector data in smaller containers from the server that can be stitched together when rendered on the client side. This one of the advantages of vector tiles over other ways of transmitting vector data. Only the vector tiles with the geographic features displayed on a Web GIS application are requested from the server. However, this is not the same case for querying RDF databases. Large vector data sets are likely to have large attributes datasets results from SPARQL queries.

Additionally, prior knowledge of the contents of the RDF databases would have been beneficial for making successful queries. This is a challenging task for novice users because they mainly depend on trial and fail approach to query results from RDF stores and RDF databases restrict heavy quires and cancel long running queries. However even though for experienced users queries with large data results can still be a challenge hence it may be beneficial to develop ontologies that can be assigned to a collection of persistent identifiers that link to geographic features in a certain vector tile. This can be developed as a pyramid scheme for attribute data linked to pyramid schemes of vector tiles.

[28] https://www.w3.org/TR/tabular-data-primer/
[29] https://www.rd-alliance.org/group/data-fabric-ig/outcomes/persistent-identifiers-consolidated-assertions
[30] https://www.iso.org/iso-3166-country-codes.html

5.3 Framework key

This study used the standardised geographic identifiers, the ISO 3166-1 numeric standard codes, as framework keys. Using the ISO standardised geographic identifiers ensured that the identifiers in the heterogenous datasets were uniform. Inconsistences in naming format used for geographic identifiers was a major challenge in the impact analysis of TJS for Statistic Netherlands research. The authors observed that some data providers would put string characters in front of the identifiers such that statistic data would have the identifier named “GM0307” vs “0307” for spatial data (Bresters et al., 2016). This resulted in many unmatched geographic identifiers.

In the study by the European and Global Forum for Geography and Statistics (EFGS) and Eurostat, the researchers proposed a new naming system for census grids after discovering that; the census grids names were not unique for the whole spatial data set. The study region used for this study spanned across several countries within Europe (EFGS and Eurostat, 2019). As a results census grid identifier that were unique within a country but could be repeated in another country. For a successful joining operation is important for geographic identifiers to be the uniformly named and unique for all input datasets.

Since this study used persistent identifiers of the standardised ISO 3166-1 numeric code there were no discrepancies in the labels of framework keys in all datasets. However, there were issues when GeoJSON features had null values for the geographic identifier field. For instance, small island countries, Antarctica and some countries do not have ISO 3166-1 numeric codes assigned to them. Therefore, GeoJSON features with a missing or unmatched geographic identifier were discarded. This was a problem when rendering the results on the OpenLayers Web Application. Missing geometry disrupted the rendering of geometric features on Web Application. There were large portions of missing geometry in the final map. The solution to this problem was either removing features with missing ISO 3166-1 numeric code or inserting mock ISO 3166-1 numeric codes before generation of vector tiles. It was vital to retain all geometry information of all geographic features in the vector tiles.

5.4 Table Joining Service Implementation

This study used a SOA approach for joining attribute data with geospatial data. The software components in the prototype were designed to be autonomous. Access and interaction between software components was through. TMS and TJS API RESTful interfaces. The incentive behind developing the OCG TJS standard was to introduce a Web service that brings together attribute data and geospatial data located on different domains. This study was able to implement a TJS protype that can retrieve attribute data in a CSV format and vector tiles encoded in GeoJSON using URLs via HTTP.

In the OGC Testbed-13 vector tiles engineering report they gave three options for handling attribute data for vector tiles (Ingensand, 2019). One of the proposed options was to store attribute data and vector tiles in separate containers; then make use of Web service to join the two datasets based on geographic features identifiers whenever it is required. The main advantage for this structure would be vector tiles would become more light making their transmission and

loading rate faster. The results of this thesis show that OGC TJS can be employed for the task of joining attribute data and vector tiles stored in separate containers. Similar studies by Hong and Lin, 2005; Bresters et al., 2016 and EFGS and Eurostat, 2019 also used the same approach of developing prototypes based on the concept of a SOA. Data providers would provide information on now to access their datasets and geographic identifiers would become the crucial factor for data harmonisation.

6. Conclusion and Future Work

The specific objectives of this research were to:

1. Examine the possibility of using cached vector tiles as a geospatial framework to be integrated with attribute data by means of a Table Joining Service.

2. Develop a prototype implementation to be used as a tool for integrating spatial data and attribute data.

3. Demonstrate through a simple Web map application the results of objective 1 using the prototype developed in objective 2.

The result of this study demonstrates that cached vector tiles can successfully be used as a spatial framework data for an OCG Table Joining Service. A TJS API prototype was developed that can access attribute data in CSV format and GeoJSON encoded vector tiles. The TJS API also provide a function that can join the two datasets based on a common geographic Identifier. This study also used some experimental data to demonstrate how the TJS API can be used to integrate attribute data with geographic features distributed as vector tiles. The results were rendered on an OpenLayers Web Application.

Previously the main source and means of transferring geospatial data for the TJS joining operation was OGC WFS. However, WFS is not very efficient with high volume data as it can be cumbersome making it slow to download and transfer. These concerns were also raised by studies done by EFGS and Eurostat, 2019 and Bresters et al., 2016. In both studies the researchers used WFS as source of geospatial data for TJS joining operation and, also as a by-product of TJS joining that could be assessed and displayed by any WebGIS client.

Vector tiles have not received much attention yet; mostly because vector tilling technology is still new (Netek et al., 2020) , there is lack of open source implementations (Balog and Houtmeyers, 2017) and there is no standardised approach of creating and publishing vector tiles (Ingensand, 2019). However, the future of vector tiles as a way of delivering web maps is very promising. The OGC is currently underway to adopt vector tilling as an OGC standard.

Besides the limitations brought by using WFS for TJS there are still issues with the TJS standard itself. Generally, there have been very few studies that have explored the feasibility of using TJS as a tool for integration of statistical and geospatial data on the Web. Moreover, there are very few TJS software implementations both open source and proprietary despite the fact the concept of TJS has existed since 2004, at first as the OCG Geolinking Service (GLS) then later as Table Joining Service an improved version of GLS in 2010.

The reason could be TJS enforces use of a GDAS data format for encoding attribute data (Aarnio and Reini, n.d.). The familiar format for publishing attribute data on the Web is CSV according to W3C [31]. Using GDAS introduces an extra complexity of developing a software that converts attribute data from already existing in formats to GDAS

[31] https://www.w3.org/2013/05/lcsv-charter.html

(Grothe and Brentjens, 2013). The National Land Survey of Finland is currently working on the revision of the TJS addressing some of these concerns. Some of the improvements will include defining TJS as a RESTful service, supporting more input and output formats and services and support use of persistent identifiers for all references to geospatial and attribute data.

7. Bibliography

Aarnio, T., Reini, J., n.d. OGC Table Joining Service standard revision and Oskari 5.

Abdalla, R., Esmail, M., 2018. WebGIS for Disaster Management and Emergency Response. Springer.

Akhtar, U., Sant'Anna, A., Jihn, C.-H., Razzaq, M.A., Bang, J., Lee, S., 2020. A cache-based method to improve query performance of linked Open Data cloud. Computing 102, 1743–1763. https://doi.org/10.1007/s00607-020-00814-9

Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2013. Web Services: Concepts, Architectures and Applications. Springer Science & Business Media.

Antoniou, V., Morley, J., Haklay, M. (Muki), 2009. Tiled Vectors: A Method for Vector Transmission over the Web, in: Carswell, J.D., Fotheringham, A.S., McArdle, G. (Eds.), Web and Wireless Geographical Information Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 56–71. https://doi.org/10.1007/978-3-642-10601-9_5

Ashton Acton, 2013. Internet Protocols—Advances in Research and Application: 2013 Edition. ScholarlyEditions.

Balog, D., Houtmeyers, R., 2017. Testbed-12 Vector Tiling Implementation Engineering Report [WWW Document]. URL https://docs.ogc.org/per/16-067r4.html (accessed 7.23.20).

Billen, R., Joao, E., Forrest, D., 2006. Dynamic and Mobile GIS: Investigating Changes in Space and Time. CRC Press.

Bocher, E., Ertz, O., 2018. A redesign of OGC Symbology Encoding standard for sharing cartography. PeerJ Computer Science 4, e143. https://doi.org/10.7717/peerj-cs.143

Bresters, P., Nl, S., van Oirschot, H.K., Nl, S., Fokke, E., Nl, S., Venema, J., Brentjens, T., Grothe, M., van Pelt, B., Hogeboom, J., Kruse, D., 2016. Impact analysis Table Joining Service 35.

Burghardt, D., Duchêne, C., Mackaness, W., 2014. Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation. Springer.

Cammack, R.G., 2007. Cartographic Approaches to Web Mapping Services, in: Cartwright, W., Peterson, M.P., Gartner, G. (Eds.), Multimedia Cartography. Springer, Berlin, Heidelberg, pp. 441–453. https://doi.org/10.1007/978-3-540-36651-5_31

Davis, C.A., Kimo, Y.J., Duarte-Figueiredo, F.L.P., 2009. OGC Web Map Service implementation challenges for mobile computers, in: 2009 17th International Conference on Geoinformatics. Presented at the 2009 17th International Conference on Geoinformatics, pp. 1–6. https://doi.org/10.1109/GEOINFORMATICS.2009.5293410

EFGS and Eurostat, 2019. Automated Linking of SDMX and OGC Web Services [WWW Document]. URL https://www.globalhealthlearning.org/gheltaxonomy/term/1177 (accessed 7.11.20).

Fensel, D., Şimşek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., Wahler, A., 2020. Knowledge Graphs: Methodology, Tools and Selected Use Cases. Springer Nature.

Fujimura, H., Sanchez, O., Ferreiro, D., Kayama, Y., Hayashi, H., Iwasaki, N., Mugambi, F., Obukhov, T., Motojima, Y., Sato, T., 2019. DESIGN AND DEVELOPMENT OF THE UN VECTOR TILE TOOLKIT. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4/W14, 57–62. https://doi.org/10.5194/isprs-archives-XLII-4-W14-57-2019

Gaffuri, J., 2012. Toward Web Mapping with Vector Data, in: Xiao, N., Kwan, M.-P., Goodchild, M.F., Shekhar, S. (Eds.), Geographic Information Science, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 87–101. https://doi.org/10.1007/978-3-642-33024-7_7

García, R., Castro, J.P. de, Verdú, E., Verdú, M.J., Regueras, L.M., 2012. Web Map Tile Services for Spatial Data Infrastructures: Management and Optimization. Cartography - A Tool for Spatial Analysis. https://doi.org/10.5772/46129

Gillies, S., Butler, H., Daly, M., Doyle, A., Schaub, T., 2016. The GeoJSON Format [WWW Document]. URL https://tools.ietf.org/html/rfc7946 (accessed 9.8.20).

Gratier, T., Spencer, P., Hazzard, E., 2015. OpenLayers 3 : Beginner's Guide. Packt Publishing Ltd.

Grothe, M., Brentjens, T., 2013. Joining tabular and geographic data – Merits and possibilities of the Table Joining Service 53.

Hong, J.H., Lin, S.Y., 2005. Web-based thematic map service in openGIS environment.

Ingensand, J. (Ed.), 2019. OGC Testbed-13: Vector Tiles Engineering Report 163.

Ingensand, J., Nappez, M., Moullet, C., Gasser, L., Ertz, O., Composto, S., 2016. Implementation of Tiled Vector Services: A Case Study, in: SDW@GIScience.

Jiang, Z., Shekhar, S., 2017. Spatial Big Data Science: Classification Techniques for Earth Observation Imagery. Springer International Publishing. https://doi.org/10.1007/978-3-319-60195-3

Krafzig, D., Banke, K., Slama, D., 2005. Enterprise SOA: Service-oriented Architecture Best Practices. Prentice Hall Professional.

Krewinkel, A., Sünkler, S., Lewandowski, D., Finck, N., Tolg, B., Kroh, L., Schreiber, G., Fritsche, J., 2015. Lebensmittelkontrolle 2.0. Journal für Verbraucherschutz und Lebensmittelsicherheit 10. https://doi.org/10.1007/s00003-015-1000-6

Kyriakidis, P., Hadjimitsis, D., Skarlatos, D., Mansourian, A., 2019. Geospatial Technologies for Local and Regional Development: Proceedings of the 22nd AGILE Conference on Geographic Information Science. Springer.

Li, L., Hu, W., Zhu, H., Li, Y., Zhang, H., 2017. Tiled vector data model for the geographical features of symbolized maps. PLoS One 12, e0176387–e0176387. https://doi.org/10.1371/journal.pone.0176387

Li, S., Dragicevic, S., Castro, F.A., Sester, M., Winter, S., Coltekin, A., Pettit, C., Jiang, B., Haworth, J., Stein, A., Cheng, T., 2016. Geospatial big data handling theory and methods: A review and research challenges. ISPRS Journal of Photogrammetry and Remote Sensing, Theme issue "State-of-the-art in photogrammetry, remote sensing and spatial information science" 115, 119–133. https://doi.org/10.1016/j.isprsjprs.2015.10.012

Lupp, M., 2008. OGC Web Services, in: Shekhar, S., Xiong, H. (Eds.), Encyclopedia of GIS. Springer US, Boston, MA, pp. 799–800. https://doi.org/10.1007/978-0-387-35973-1_903

Mahmud, S.M.H., Hossin, Md.A., Hasan, Md.R., Jahan, H., Noori, S.R.H., Ahmed, Md.R., 2020. Publishing CSV Data as Linked Data on the Web, in: Singh, P.K., Panigrahi, B.K., Suryadevara, N.K., Sharma, S.K., Singh, A.P. (Eds.), Proceedings of ICETIT 2019, Lecture Notes in Electrical Engineering. Springer International Publishing, Cham, pp. 805–817. https://doi.org/10.1007/978-3-030-30577-2_72

Martinelli, L., Roth, M., 2015. Vector Tiles from OpenStreetMap (srp). HSR Hochschule für Technik Rapperswil.

Masó, J., Pomakis, K., Julià, N., 2010. OpenGIS® Web Map Tile Service Implementation Standard, v. 1.0. 0, Open Geospatial Consortium Inc. Document reference number 07-057r7, 2010.

Masse, M., 2011. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O'Reilly Media, Inc.

Netek, R., Masopust, J., Pavlicek, F., Pechanec, V., 2020. Performance Testing on Vector vs. Raster Map Tiles—Comparative Study on Load Metrics. ISPRS International Journal of Geo-Information 9, 101. https://doi.org/10.3390/ijgi9020101

OGC Vector Tiles Pilot: WFS 3.0 Vector Tiles Extension Engineering Report [WWW Document], 2019. URL https://docs.ogc.org/per/18-078.html (accessed 10.5.20).

Peterson, M.P., 2012. Online Maps with APIs and WebServices. Springer Science & Business Media.

Sambells, J., Purvis, M., Turner, C., 2007. Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional. Apress.

Shang, X., 2015. A Study on Efficient Vector Mapping With Vector Tiles Based on Cloud Server Architecture. http://dx.doi.org/10.11575/PRISM/25046

Tiwari, A., Jain, D.K., 2014. GEOSPATIAL FRAMEWORK FOR DENGUE USING OPEN SOURCE WEB GIS TECHNOLOGY [WWW Document]. URL /paper/GEOSPATIAL-FRAMEWORK-FOR-DENGUE-USING-OPEN-SOURCE-Tiwari-Jain/c47320c957aa9adf2a07d8398c49f7cd10cce85f (accessed 10.8.20).

Veenendaal, B., Brovelli, M.A., Li, S., 2017. Review of Web Mapping: Eras, Trends and Directions. ISPRS International Journal of Geo-Information 6, 317. https://doi.org/10.3390/ijgi6100317

VoPham, T., Hart, J.E., Laden, F., Chiang, Y.-Y., 2018. Emerging trends in geospatial artificial intelligence (geoAI): potential applications for environmental epidemiology. Environmental Health 17, 40. https://doi.org/10.1186/s12940-018-0386-x

Wan, L., Huang, Z., Peng, X., 2016. An Effective NoSQL-Based Vector Map Tile Management Approach. ISPRS International Journal of Geo-Information 5, 215. https://doi.org/10.3390/ijgi5110215

Warf, B., 2018. The SAGE Encyclopedia of the Internet. SAGE.

Zouhar, F., Senner, I., 2020. Web-Based Visualization of Big Geospatial Vector Data, in: Kyriakidis, P., Hadjimitsis, D., Skarlatos, D., Mansourian, A. (Eds.), Geospatial Technologies for Local and Regional Development, Lecture Notes in Geoinformation and Cartography. Springer International Publishing, Cham, pp. 59–74. https://doi.org/10.1007/978-3-030-14745-7_4

Appendix 1

Script for the Web Server.

```
import flask
from flask import Flask
# pip install sparqlwrapper
# https://rdflib.github.io/sparqlwrapper/

import sys
from SPARQLWrapper import SPARQLWrapper, JSON
import pandas as pd

un_df = pd.read_csv("static\sample-csv.csv")
# dfcolumns = ['UN_A3', 'country_name', 'sum']

# df = df.reindex(columns=dfcolumns)
# define the columns with key to be of same data type
# df['UN_A3'] = df['UN_A3'].astype(int)
# df = df[['UN_A3', 'country_name', 'sum']]
# print(un_df)

endpoint_url = "https://query.wikidata.org/sparql"

query = """#Find ISO 3166-1 alpha-2 country codes
SELECT distinct ?countryLabel ?UN_A3 ?ISO_A2?Population
WHERE
{
  ?country wdt:P299 ?UN_A3 .
  ?country wdt:P297 ?ISO_A2 .
  ?country wdt:P1082 ?Population.
        SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en" }
} order by ?ISO_A2"""


def get_results(endpoint_url, query):
    user_agent = "WDQS-example Python/%s.%s" % (sys.version_info[0], sys.version_info[1])
    # TODO adjust user agent; see https://w.wiki/CX6
    sparql = SPARQLWrapper(endpoint_url, agent=user_agent)
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    return sparql.query().convert()


results = get_results(endpoint_url, query)
# Print results as dataframe
results_df = pd.io.json.json_normalize(results['results']['bindings'])
# wiki_df = results_df[['countryLabel.value', 'UN_A3.value', 'ISO_A2.value', 'Population.value']]

wiki_df = results_df[['UN_A3.value', 'Population.value']]

```

```
48.  # Convert column names
49.  wiki_df.columns = ["UN_A3_1", "Population"]
50.  #print(wiki_df)
51.  wiki_df_columns = ["UN_A3_1", "Population"]
52.  un_df_columns = ["UN_A3", "Country_name",'GDP(2005)']
53.  un_df = un_df.reindex(columns=un_df_columns)
54.  wiki_df = wiki_df.reindex(columns=wiki_df_columns)
55.  un_df['UN_A3'] = un_df['UN_A3'].astype(int)
56.  wiki_df['UN_A3_1'] = wiki_df['UN_A3_1'].astype(int)
57.
58.  result = un_df.join(wiki_df.set_index('UN_A3_1'),on=["UN_A3"])
59.  #print(result)
60.  result.to_csv(r'C:\Users\Sharon\TJS\TJSpythonProject\static\un_wiki.cs
     v', index = False)
61.  # mock_server = 'http://localhost:8000'
62.  app = Flask(__name__)
63.
64.  # Get url for a static file.
65.  with app.test_request_context():
66.      att_uri = flask.url_for("static", filename="sample-csv.csv")
67.      # print(att_uri)
68.
69.
70.  @app.route('/')
71.  def index():
72.      return 'TJS'
73.
74.
75.  app.run(debug=True, port=8000)  # run app in debug mode on port 5000
```

Appendix 2

Script for the Table Joining Server and TJS API

```
import geopandas as gpd
import pandas as pd
from flask import Flask, request


app = Flask(__name__)


def get_frameworkdata(vtc_url):
    gdf = gpd.read_file(vtc_url)
    return gdf


 def get_attributedata(attribute_url):
    df = pd.read_csv(attribute_url)
    return df


 def get_frameworkkey(frameworkkey, attribute1, attribute2):
    framework_key = str(frameworkkey)
    attribute_1 = str(attribute1)
    attribute_2 = str(attribute2)
    return [framework_key, attribute_1, attribute_2]

 @app.route('/')
 def index():
    return 'TJS'


 @app.route('/tjs/api', methods=['GET'])
 def join_data():
    vtc_url = request.args.get('vtc_url')
    attribute_url = request.args.get('attribute_url')
    framework_key = request.args.get('framework_key')
    attribute1 = request.args.get('attribute1')
    attribute2 = request.args.get('attribute2')

    gdf = get_frameworkdata(vtc_url)
    adf = get_attributedata(attribute_url)
    keys = get_frameworkkey(framework_key, attribute1, attribute2)
    gdf_columns = ['UN_A3', 'geometry']
    adf_columns = keys
    adf = adf.reindex(columns=adf_columns)
    gdf = gdf.reindex(columns=gdf_columns)
    adf['UN_A3'] = adf['UN_A3'].astype(int)
    gdf['UN_A3'] = gdf['UN_A3'].astype(int)
    geometry = gdf[['geometry', 'UN_A3']]
    attributes = adf[keys]
    geometry = geometry.merge(attributes,
on='UN_A3').reindex(gdf.index)
```

```
50.     geojson = geometry.to_json()
51.     return geojson
52.
53.
54. app.run(debug=True, port=5000)  # run app in debug mode on port 5000
55.
```

Appendix 3

Script for the OpenLayers Web Application.

```
1.  <!DOCTYPE html>
2.  <html>
3.   <head>
4.     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
5.     <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
    scalable=0">
6.     <script src="http://code.jquery.com/jquery-latest.js"></script>
7.     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/2.2.1/js/bootstrap.min.js"></script>
8.     <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.12/OpenLayers.js"></script>
9.     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/2.2.1/css/bootstrap.min.css">
10.    <style>
11.                body {
12.          padding-top: 60px;
13.          padding-bottom: 40px;
14.        }
15.        .map {
16.          height: 512px;
17.          background-color: #eee;
18.                          border: 1px solid #CCCCCC;
19.        }
20.
21.
22.    </style>
23.    </head>
24.   <body ">
25.    <div class="navbar navbar-inverse navbar-fixed-top">
26.      <div class="navbar-inner">
27.        <div class="container-fluid">
28.          <a class="brand" href="/">Web Thematic Map created by OGC TJS</a>
29.        </div>
30.      </div>
31.    </div>
32.    <div class="container-fluid">
```

```
33.      <div class="row-fluid">
34.        <div class="span5">
35.          <div id="map" class='map'>
36.                                    </div>
37.        </div>
38.        <div class="span7">
39.                                    <div>
40.            <table>
41.                                      <tr>
42.                                        <td><img src="data/key.svg"/></td>
43.
44.                                      </tr>
45.                                     </table>
46.          <div id="metroImage"></div>
47.                 </div>
48.        </div>
49.      </div>
50.    </div>
51.
52.
53.   <script src="main.js"></script>
54.   </body>
55. </html>
```

```
1.  import 'ol/ol.css';
2. import {Vector} from 'ol/source';
3. import {GeoJSON} from 'ol/format';
4. import Map from 'ol/Map';
5. import View from 'ol/View';
6. import VectorLayer from 'ol/layer/Vector';
7. import {Fill, Stroke, Style, Text} from 'ol/style';
8. import VectorTileLayer from 'ol/layer/VectorTile';
9. import VectorTileSource from 'ol/source/VectorTile';
10.  import MVT from 'ol/format/MVT';
11.  import Layer from 'ol/layer/Layer';
12.
13.
14.  var style_simple = new Style({
15.      fill: new Fill({
16.        color: '#ADD8E6'
17.      }),
18.      stroke: new Stroke({
19.        color: '#880000',
20.        width: 1
```

```
21.      })
22.    });
23.
24.    function simpleStyle(feature) {
25.      return style_simple;
26.    }
27.
28.    var colorGradient = [
29.    'rgb(128,128,128)',
30.
   'rgb(140,81,10)','rgb(216,179,101)','rgb(246,232,195)','rgb(199,234,229
   )','rgb(90,180,172)','rgb(1,102,94)'
31.  ]
32.
33.  // map the income level codes to a colour value, grouping them
34.  var gradStyle =  function(feature, resolution) {
35.          //console.log(feature.get('sum'));
36.          console.log(feature.getId()+ ': ' +feature.get('Country_name')
   + ': ' + feature.get('GDP(2005)'));
37.      var data = feature.get('GDP(2005)');
38.      var color;
39.      if ( data < 50000 ) {
40.        color = colorGradient[6];//low value
41.      } else if ( data >= 50000 && data < 500000 ) {
42.        color = colorGradient[5];//
43.      } else if ( data >= 500000 && data < 1000000 ) {
44.        color = colorGradient[4];
45.      } else if ( data >= 1000000 && data < 3000000) {
46.        color = colorGradient[3];
47.          }  else if ( data >= 3000000 && data < 5000000) {
48.        color = colorGradient[2];
49.          }  else if ( data >= 5000000 ) {
50.        color = colorGradient[1];
51.      }
52.          else if ( data = 'null'  ) {
53.        color = colorGradient[0];
54.      }
55.      return new Style({
56.        stroke: new Stroke({
57.            color: 'black', //'rgba(255, 255, 255 ,1.0)',
58.            //lineDash: [3, 3],
59.            lineCap: 'butt',
60.            lineJoin: 'miter',
61.            width: 0.5,
62.        }),
63.        fill: new Fill({
64.          color: color
65.        }),
66.      });
67.
68.    }
69.
70.
71.   var url =
   'http://127.0.0.1:5000/tjs/api?vtc_url=http://localhost:8080/geoserver/
   gwc/service/tms/1.0.0/'+
```

```
  'countries%3Ane_110m_admin_0_countries@EPSG%3A3857@geojson/{z}/{x}/{-y}.geojson&'+
  'attribute_url=http://127.0.0.1:8000/static/sample-csv.csv&'+
  'framework_key=UN_A3&attribute1=Country_name&attribute2=GDP(2005)'


    const layer = new VectorTileLayer({
            //style:simpleStyle,
            style: gradStyle,
     source: new VectorTileSource({
         attributions: '',
         format: new GeoJSON(),
         maxZoom: 19,
                url: url,


    //'http://localhost:8080/geoserver/gwc/service/tms/1.0.0/' +
   //'countries:ne_110m_admin_0_countries@EPSG%3A900913@geojson/{z}/{x}/{-y}.geojson',
        tileLoadFunction: function(tile, url) {
                 tile.setLoader(function(extent, resolution, projection) {
                         fetch(url).then(function(response) {
                                response.text().then(function(data) {
                                        const jsons = JSON.parse(data);
                                        const format = tile.getFormat();
                                        console.log(data);
    tile.setFeatures(format.readFeatures(data));});
                                 });
                          });
                          },


     })
  });


 var map = new Map({

  view: new View({
    center: [0, 0],
    zoom: 2,
        maxZoom:20
  }),
  layers:[layer],
  target: 'map'
});
```

```
1. {
2.   "name": "mywebmap2",
3.   "version": "1.0.0",
4.   "description": "",
5.   "main": "main.js",
6.   "scripts": {
7.     "test": "echo \"Error: no test specified\" && exit 1",
8.   "start": "parcel index.html",
9.     "build": "parcel build --experimental-scope-hoisting --public-url .
   index.html"
10.    },
11.    "author": "",
12.    "license": "ISC",
13.    "dependencies": {
14.      "@geoext/geoext": "^3.2.0",
15.      "@terrestris/basigx": "^2.0.2",
16.      "ol": "^6.4.3"
17.    },
18.    "devDependencies": {
19.      "parcel-bundler": "^1.12.4"
20.    }
21.  }
```