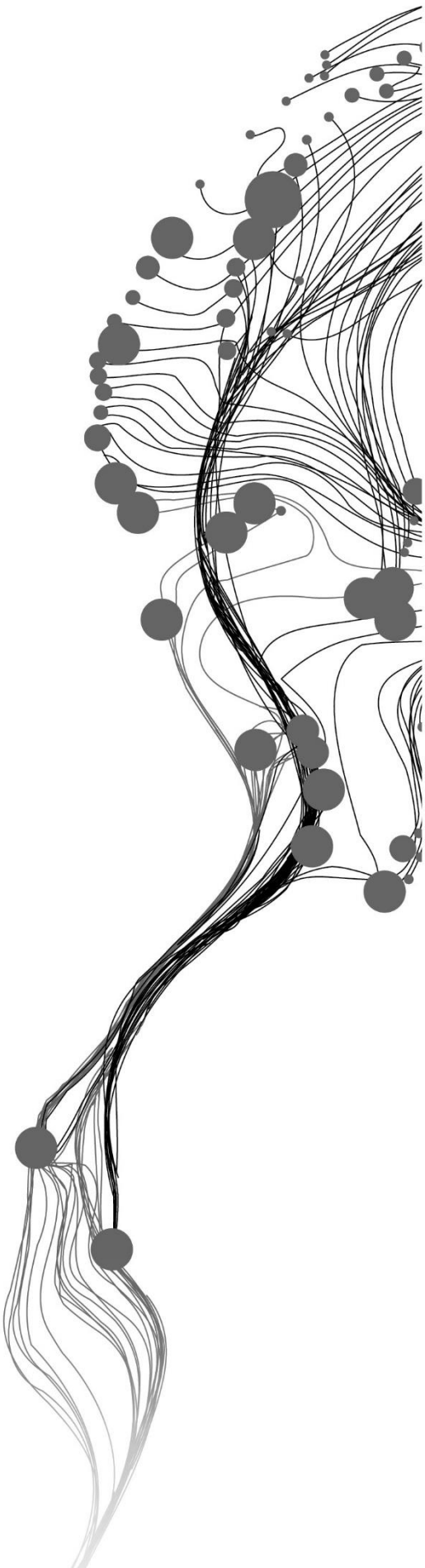
ARCHITECTURAL PATTERN FOR DESIGNING GEOSPATIAL WEB APPLICATIONS

JIARONG LI
AUGUST, 2021

SUPERVISORS:
DR. J.M. MORALES GUARIN
DR.IR. R.A. DE BY



ARCHITECTURAL PATTERN FOR DESIGNING GEOSPATIAL WEB APPLICATIONS

JIARONG LI
Enschede, The Netherlands, AUGUST, 2021

Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:
DR. J.M. MORALES GUARIN
DR.IR. R.A. DE BY

THESIS ASSESSMENT BOARD:
PROF.DR. R. ZURITA MILLA (Chair)
DR. IR. E.J.A. FOLMER (External member, Dutch Kadaster)
DR. F.-B. MOCNIK (Procedural advisor)

ABSTRACT

The development of web technologies drives the growth and evolution of geospatial web applications (geo-web apps). The combination of web and Geographic Information Science (GIS), i.e., WebGIS, is in rapid development on technologies like spatial data storage and web mapping. However, there is a lack of research on geo-web app in a conceptual level. There are recurrent problems in developing of geo-web apps. To use concrete solutions to address these problems can suffer from the dependency between system elements. It is of significant value to find a reusable solution that is independent from the platform, programming language, .etc.

Design pattern is a scheme to introduce reusable approaches to addressing recurrent problems in software design. A fundamental of the pattern approach is built by the 23 GoF patterns. The architectural pattern is a subset of patterns covering the concerns of a whole application. The applying of architectural patterns, e.g., Model-View-Controller (MVC) achieves a high level of reusability in the system design. This approach can be employed to the design of geo-web apps.

This research describes a study, to design an architectural pattern for geo-web apps from analyzing the problems. The common concerns of developing a geo-web app are discussed, using commonality and variability analysis. Based on these concerns, the pattern elements are determined with a structure from specified architectural viewtypes. Then a formal representation of the architectural pattern is given, as an instruction for implementation. Finally, there is a sample implementation presented to validate the pattern, and a discussion on the consequence of the pattern.

Keywords: Geo-web app, Reuse, Architectural Pattern, Design Pattern, Architectural View, Commonality and Variability Analysis, Interactive Web App

ACKNOWLEDGEMENTS

I would like to express my gratitude to people who helped me during the research process.

Firstly, I would like to thank ITC faculty and my supervisor, dr. J.M. Morales Guarin and dr.ir. R.A. de By. The study period at ITC benefits me a lot, in professional skills and soft skills. In terms of research and thesis writing, the suggestions and feedback from my supervisors help a lot in improving the product.

Secondly, I would like to thank dr. ir. E.J.A. Folmer, and other members in Kadaster Data Science Team, for giving me the opportunity to take the internship, which provides resources of dataset, tools, and reference applications for this research. Also this internship gave me much support to make me more confident.

Next, I would like to thank my parents and my friends in China. They gave me quite a lot of support. Thanks for trusting me and helping me.

Then, I would like to thank my classmates, Shichen, Xiaoyu and Morteza, for the communications on the problems about the thesis. Their suggestions contribute to the research processing.

Finally, I would like to thank my boyfriend Fangyuan. We did not have a chance to come back to our hometown in these two years. Fortunately that we have each other.

Thanks again to all who helped me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVEATIONS

GIS	Geographic Information System
Geo-Web App	Geospatial Web Application
OO	Object-Oriented
GRASP	Genberal Responsibility Assignment Software Patterns
GoF	Gong of Four
MVC	Model-View-Controller
WWW	World Wide Web
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
AJAX	Asynchronous JavaScript and XML
OGC	Open Geospatial Consortium
JSON	JavaScript Object Notation
GeoJSON	Geographic JavaScript Object Notation
GML	Geography Markup Language
KML	Keyhole Markup Language
KMZ	Keyhole Markup Language files when compressed
SVG	Scalable Vector Graphics
W3C	World Wide Web Consortium
TIFF	Tag Image File Format
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
WKT	Well known text
WKB	Well-Known Binary
WMS	Web Map Service
WFS	Web Feature Service
WPS	Web Processing Service
WMTS	Web Map Tile Service
WCS	Web Coverage Service
NETCDF	Network Common Data Form
SOA	Service-Oriented Architecture
API	Application Programming Interface
UML	Unified Modeling Language
CRC	Coronavirus Resource Center
BRT	Basisregistratie Topografie
URI	Uniform Resource Identifier
CVA	Commonality and Variability Analysis
SSGIS	Self-Service GIS
SPARQL	SPARQL Protocol and RDF Query Language
PDOK	Publieke Dienstverlening Op de Kaart
BAG	Basisregistratie Adressen en Gebouwen
BGT	Basisregistratie Grootschalige Topografie
UI	User Interface

1. INTRODUCTION

1.1. Background and Motivation

A geospatial application, which is a kind of software that deals with geospatial data, allows users to explore and integrate geospatial data and employs geoprocessing functions. Geospatial applications distinguish from other types of applications in considering geospatial features. Many purposes, such as education, activity planning, and entertainment are expected to be served in this kind of application(Simon & Fröhlich, 2007). The development of geospatial applications can be based on some existing Geographic Information System(GIS) developing frameworks or be created independently.

The growth of web technology drives the wide use of web applications. Users can get access to the services of this kind of application through a web browser. Compared with desktop software, a web application shows advantages such as independence of platform and eliminating space limitations. As a combination of GIS and web technology, geospatial web application(geo-web app) is in rapid development for its convenience in publishing and visualizing geospatial data.

According to the development of geo-web apps, there are recurrent problems in conceptual design, resulting from the complexity of geospatial data and the unique user requirements. The reusable approach to solve these recurrent problems for building a geo-web app is of significant importance.

A design pattern is a strategy to present reusable solutions to common problems in software design based on object-oriented technology, which is a widely used programming paradigm that can improve the reusability and flexibility of software. And an architectural pattern describes the architecture of the whole application in a universal way. The use of patterns in software engineering enables the reuse of successful architectures and designs(Gamma, Helm, Johnson, & Vlissides, 1995). It also benefits teamwork in describing solutions in a standard way.

As discussed above, the use of patterns can contribute to the developing of geospatial applications at the conceptual level. However, the innovative patterns proposed in the GIS domain and research in applying existing patterns to developing GIS applications are not sufficient. There are explorations of applying design patterns to some aspects of developing a geospatial application, such as map display and spatial data management(Coetzee & Rautenbach, 2018), but there still exists a research gap in resolving the common problems in geo-web apps developing by design pattern approach, and providing a solution in architectural level.

1.2. Problem Statement

In this MSc research, I connect the concept of patterns to the developing of geo-web apps, by proposing an architectural pattern as a reusable template. The research focuses on the exploration of common concerns, the adaptation of pattern approach to manipulating these concerns, and the implementation methods of this approach.

1.3. Research Identification

1.3.1. Research Objective

The overall objective of this research is to design an architectural pattern for developing geo-web apps. It can be divided into three sub-objectives as follows:

1. To identify the common concerns in developing a geo-web app.

2. To design an architectural pattern for addressing the common concerns, as a clean template for building a geo-web app.
3. To implement the pattern by developing an application based on this template.

1.3.2. Research Questions

1. **To identify the common concerns in developing a geo-web app.**
   i. What are the desired requirements and functionalities of a geo-web app?
   ii. What are the common implementation elements of a geo-web app?

2. **To design an architectural pattern for addressing the common concerns, as a clean template for building a geo-web app.**
   i. How to construct the implementation elements for establishing an architectural pattern?
   ii. What are the required architectural elements, relations between them and constraints on them?
   iii. What is the appropriate approach to representing the architectural pattern?

3. **To implement the pattern by developing an application based on this template.**
   i. What is the workflow to develop an application applying the architectural pattern?
   ii. What are the required types of techniques in the implementation, and what are the principles to select the techniques?
   iii. What is the consequence of applying the architectural pattern?

1.4. Structure of the Thesis

Chapter 1 provides an introduction to the background and motivation of the research. And the research problem is defined, followed by the statement of research objectives and research questions.

Chapter 2 gives a literature review as the basis of the research. The definition of concepts about pattern and geo-web app is introduced. The significance of design pattern is then explained. And in terms of architectural pattern, the concept of architectural view is stated, as a fundamental of the methodology to construct a pattern. The evolution of geo-web apps is reviewed, with an introduction to WebGIS technology and architecture.

Chapter 3 explains the methodology for steps of the research. The research framework is given as an overview of the methodology. The commonality and variability analysis to retrieve the common concerns and implementing elements, the architectural view types to construct the pattern and the implementation process are discussed respectively.

Chapter 4 presents the outcome of sub-objective1. The process and result to reach the common concerns of geo-web apps are explained. There is a discussion on common user requirements and functionality, by analysis on typical applications. It is followed by the identification of concepts and implementing elements reached by the commonality and variability analysis.

Chapter 5 presents the result of sub-objective2. It explains the process to define the architectural pattern elements, based on the selected architectural viewtypes and reference patterns, and the analysis on problem domain in the previous chapter. A formal representation of the architectural pattern is given.

Chapter 6 presents the result of sub-objective3. The implementation of Pandviewer application by using the architectural pattern is introduced, following the workflow discussed in Chapter 3. And there is a review on the implementation of the pattern.

Chapter 7 provides the conclusion and recommendation of the research. The answers to the research questions are given, followed by the contributions and limitations of the research and recommendations for future work.

2. LITERATURE REVIEW

This chapter provides an overview of the relevant research on this topic. It starts from the concepts of object-oriented methodology and design patterns. The introduction to architectural patterns is then given. Finally there is an overview of geo-web apps, on the evolution, relevant technology and architecture types.

2.1. Object Oriented Methodology and Design Patterns

This section briefly introduces the concepts of object-oriented design (OO design) and design patterns. Definitions of object-oriented design, reuse, and design pattern are given, with discussions on the previous work in this domain.

2.1.1. Object-oriented design

The object-oriented methodology describes a system with objects and classes. It is proved to be a good solution for the design of a system with high complexity of data and relationships, as the reusable, modular, modifiable software can be obtained(Gordillo et al., 1999).
There are basic concepts for an OO design, which are Encapsulation, Inheritance, Polymorphism, Aggregation/Composition, Interface/Implementation, Abstraction(Larman, 2004). These concepts are described as follows:

- **Encapsulation** - The implementation details and internal state of an object is hidden, and the only way to access the object data is by its interface. By encapsulation the private data can be protected, and the complexity of object details can be separated.
- **Inheritance** - A child class gets all the properties and behaviors of the parent class, and it can have its additional properties and behaviors.
- **Polymorphism** - The class inherited from another class can stand for the superclass. And the subclass can have many forms.
- **Aggregation/Composition** - The aggregation represents a 'has a' relation between classes, and composition is a strong type of aggregation, in which the part cannot live without the whole.
- **Interface/Implementation** - The interface defines how users interact with a class, and the implementation is to fulfill the methods defined by the interface.
- **Abstraction** - Abstraction is the methodology to ignore the details and pay attention to important aspects. It is the essential method to find classes when designing a system.

The concepts define the basic rules of OO design, and there are General Responsibility Assignment Software Patterns (GRASP) principles describing methods to reach a good design. The overview of these principles are as follows:

- **Creator** – It defines which class is responsible for creating an instance of a class, according to the relations between classes.
- **Information Expert** – It defines the principle to assign responsibilities to objects, that the class which owns the information to fulfill the responsibility should take the responsibility. Information Expert ensures encapsulation and can distribute behaviors among classes.
- **Low Coupling** – It is a principle to remain low coupling between modules, in order to reduce dependency and increase reuse.
- **Controller** – It defines that the first object to receive user controls in a system should be a Façade Controller or a Use Case Controller.

- **High Cohesion** – It is a principle to remain a high cohesion of elements within a module, to keep objects focused, understandable and manageable.
- **Polymorphism** – It defines the principle to assign responsibility using polymorphic operations when the behaviors vary by type, instead of using condition logic.
- **Indirection** – To avoid direct coupling between objects, an intermediate object can be used, which also ensures reusability.
- **Pure Fabrication** - It defines another method to assign responsibilities, that is using an artificial class, which does not represent a problem domain concept. Sometimes the reusability, low coupling and high cohesion are supported by this principle.
- **Protected Variations** – It is a principle to identify the variations in a system and create a stable interface to isolate the variations.

These principles describe the OO design in an understandable way. Principles like Creator and Indirection provide suggestions and references for conceptual design of software, and High Cohesion and Low Coupling always serve as evaluation tools for a design.

2.1.2. Reuse in software design

Code reuse plays an essential role in software design as it reduces development costs and time(*Refactoring and Design Patterns*, n.d.). Object-oriented methodology introduces approaches to fulfill reuse, e.g., class inheritance. Erich Gamma described code reuse in three levels(*Artima - Erich Gamma on Flexibility and Reuse*, n.d.):

The reuse of classes is at the lowest level, that class terms such as class libraries are reused. The reuse of framework is at the highest level, which provides solutions in a large granularity. Despite the higher usability compared with class reuse, framework shows higher risk and less customizability. Design pattern, which is at the middle level, describes how classes relate to and interact with each other. It is widely used as a reusable solution for software design.

2.1.3. Design Patterns

In software engineering, a design pattern describes a reusable solution for a problem that occurs commonly in an abstract way(Shalloway & Trott, 2002). The concept of design pattern is proposed by Christopher Alexander, as a scheme put forward to solve recurrent problems. It can be used repeatedly by other people to reduce the effort to think about solutions. Gemma et al. describe the software design patterns in a systematic way, by concluding and explaining 23 GoF patterns(Gamma et al., 1995). They also classified the patterns into three categories by their purposes into creational patterns, structural patterns and behavioral patterns.

A design pattern is composed of four essential elements: the name of the pattern, the problem, the solution and consequences and tradeoffs. Gemma et al. introduced a method to describe a pattern with the following items:

- **Intent** – The purpose of the pattern.
- **Motivation** - The motivation of proposing this pattern.
- **Applicability** – When this pattern can be used.
- **Structure** – The structure of pattern concepts. It is always represented by a graphical notation.
- **Participants** - Description of each concept.
- **Collaborations** - How these concepts collaborate with each other.
- **Consequences** - The result of using this pattern.
- **Implementation** - What risks, hints, or techniques should be aware of when implementing this pattern.
- **Sample code** - The code to illustrate how to implement this pattern.
- **Related patterns** - What design patterns are related to this one closely.

Design patterns can be used to solve specific problems, in order to reduce the effort of redesign. The consequence of a design pattern is always in line with OO design principles. And knowledge from experts can be passed on to novices by the proposal of a pattern. Also the use of design patterns ensures easy documentation since it introduces common vocabulary. Finally, the system refactor benefits from design patterns.

2.2. Architectural Patterns

This section gives a literature review on architectural patterns, including the definitions of this concept and research on pattern languages to represent the pattern. And discussion on some classic architectural patterns is given.

2.2.1. Architectural pattern and architectural style

The architectural pattern is a kind of universal and high-level pattern. The difference between architectural pattern and design pattern is, architectural pattern can be used to design the architecture of an entire application, while design pattern always contributes to a part of design.
The architectural style, which also refers to a clean template of software architecture, is a similar concept with the architectural pattern. In the perspective of architectural patterns, patterns are considered as problem-solution pairs that the design problems are specified. In the perspective of architectural styles, the attention is not on a specific problem. Instead, it focuses on terms of components, connectors, and issues related to control and data flow(Shaw & Clements, 1997). Avgeriou et al. suggest that architectural pattern and architectural style are basically the same concepts, and the only difference is in description forms(Avgeriou & Zdun, 2005).

2.2.2. Pattern Languages

In order to describe and classify the architectural patterns, a pattern language can help. Unlike design patterns, architectural patterns cover concerns in an entire architecture, beyond the class structure and relations.
An architectural view is a representation of a system from the perspective of a related set of concerns, which includes the elements and relations(IEEE-SA Standards Board, 2000). The viewpoint can describe the type of elements and relations. An architectural pattern can be considered as a specialization of a viewpoint since there is a specified semantics to describe the element types, relations and constraints (Avgeriou & Zdun, 2005). The relation of architectural views, viewpoints, and patterns are presented in Figure 1.

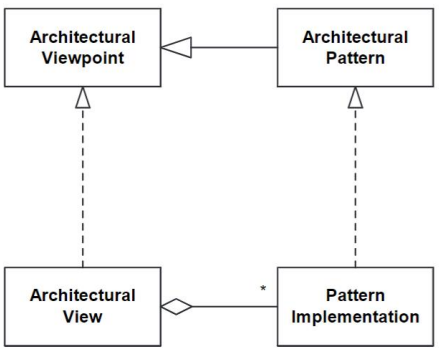


Figure 1: Architectural view, viewpoints and patterns(Avgeriou & Zdun, 2005)

Viewtype refers to the category of architectural views. Clements et al. proposed architectural viewtypes of Module, Component-and-Connector, and Allocation(Clements et al., 2003). Avferiou et al. introduced a classification method in smaller granularity, based on the Component-and-Connector viewtype(Avgeriou & Zdun, 2005). Architectural views are categorized into Layered, Data Flow, Data Cantered, Adaptation, Language Extension, User Interaction, Component Interaction and Distribution, as different perspectives of architectural patterns.

2.2.3. MVC pattern

As a commonly used pattern, Model-View-Controller (MVC) is described in this sub-section as an example of architectural patterns. It is firstly introduced in 1970 in Smarttalk-79 by Reenskaug, as a methodology to separate user interface from the programming logic(*Trygve/MVC*, n.d.). An overview of MVC pattern is described in Table 1.

Table 1: Overview of MVC pattern

Name	Model-View-Controller (MVC)
Problem	A system with multiple user interfaces should enable automatic reflection of changes on data to these interfaces. The modification on user interfaces should be easy, without influencing the application logic.
Solution	The system is divided into a Model, to manipulate data and application logic, one or multiple Views, to display the data, and a Controller, to associate the Model with Views.
Consequence	The user interface and application logic are strictly separated, which enables high interactive functionalities.

MVC pattern can be discussed under the viewtype of User Interaction View, in which the association between user interface and application logic concerns. Figure 2 illustrates an example of MVC architecture, revealing the structure of components and collaboration between components.

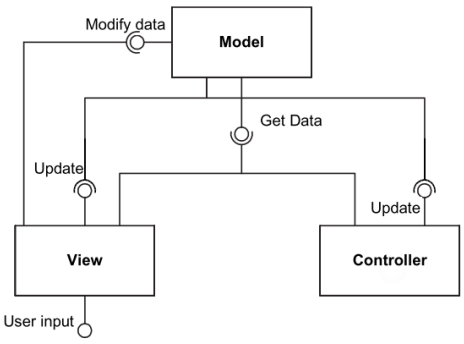


Figure 2: An Model-View-Controller example(Avgeriou & Zdun, 2005)

2.3. Geo-Web App Overview

In this section, an overview of concepts and relevant work on geo-web apps are given. There are discussions on the evolution of geo apps and the combination with web technology. WebGIS technologies relevant to spatial data and services are explained in detail, followed by a discussion on WebGIS architecture.

2.3.1. Evolution of geo-web app

The need of manipulating spatial information drives the development of GIS, which is defined as a powerful set of tools for collecting, storing, retrieving, converting and displaying spatial data from the real world(Burrough, 1986). GIS is used to answer generic location-based questions.
The rapid development of web technology brings convenience to distributing applications. The World Wide Web(WWW) is a collection of hypertext documents that are linked to other documents located on computers anywhere in the world(Ingram, 1995). And there are protocols defined by WWW to exchange files. Hypertext Markup Language(HTML) and Hypertext Transfer Protocol (HTTP) enables features like forms and applet objects. Web applications run on a web server and show results depending on the user input from a browser(Conallen, 1999). Compared with desktop applications, web applications are independent of the operation environment and easy to maintain and update. The growth of web has a significant effect on GIS, which leads to the development of the geo-web app.
The integration of GIS and web technology enables functionalities such as interactive access to geospatial data and real-time data integration(Su et al., 2000). In 1993 Putz developed an interactive web map site, as the first initiative of WebGIS. In 1999, the releasement of Web 2.0 has a major impact on the development of geo-web apps. In 2005, Asynchronous JavaScript and XML (AJAX) was introduced, which results in the launch of lots of interactive geo-web apps, e.g., Google Map. These milestones indicate the evolution of WebGIS with the growth of web technology.

2.3.2. WebGIS technology

WebGIS technology plays an important role in the design of a geo-web app. In this sub-section the spatial data encoding, spatial data store types, geospatial services are explained in detail. These technologies concern the storing and manipulating of spatial data.

2.3.2.1. Spatial data encoding

A geo-web app involves the transferring of geodata from server to client, in which process the data format is a decisive question(Luqun et al., 2002). The geodata transferred to the client can be raster or vector, and the choice makes a difference to the flexibility and performance of web mapping.
Raster data are digital images represented by grids. It usually can be displayed by a web browser directly. Vector is a format to represent geo features, with the types of point, polyline, polygon, etc. An extension in the application may be needed to display the vector data. The advantage of using raster format over that of vector in web mapping is that the rendering is less expensive computationally. However, the employing of vector format introduces more flexibilities. The raster data is always static with the boundary and appearance, while those of vector data can be dynamic, which enables some functions like highlighting a feature. It is also easier to conduct the geoprocessing on vector data according to the web mapping. In addition, the vector data fits better with the environment of relational database(*GIS File Formats - Wikipedia*, n.d.). Except for the difference in performance and operations, there is also a distinction in the way to store non-spatial information between raster and vector data. For vector data, the non-spatial information is stored as attributes of a feature. For raster data, the non-spatial information can be stored as the cell value. It makes sense to make the decision on which type of format to use in building a geo-web app based on the requirements.
Open data formats are those formats that are openly documented and have no restrictions on their use(*The Data Tier Of Your Web Mapping Architecture | GEOG 585: Web Mapping*, n.d.). It is usually maintained by a standards organization. Examples of open geodata formats are shapefiles, KML and GML. Proprietary data formats are those developed by an organization or company and with restrictions on use. The geodata in proprietary format can only work in a particular software or environment.
Open Geospatial Consortium (OGC) is an organization contributing to the open standards of geospatial content(*OGC Standards | OGC*, n.d.). The OGC standards define the standardized encoding of raster and vector data. For example, Geography Markup Language (GML) is a format to present geographical

features based on the XML grammar. It is defined by OGC as a standard in the transport and storage of geoinformation.
To give an overview of the encodings of geodata and their features, some typical formats of vector and raster data are discussed. And the properties of these encodings are introduced in Table 2 and Table 3.

Vector encoding:

- **Esri Shapefile** - A format promoted by ESRI to store geometry and attribute information of the spatial features(*ESRI Shapefile Technical Description*, 1998). It is an open format and is widely used in GIS applications. A set of shapefile contains multiple files to express the geometry, attributes and other information about the vector data.
- **Geographic JavaScript Object Notation (GeoJSON)** - A format to represent geo features based on JavaScript Object Notation (JSON). The coordinates and attributes are included in a GeoJSON file. It is widely used in web mapping since JavaScript is the default client-side language and can be understood by web browsers.
- **Geography Markup Language (GML)** - A standardized spatial data format defined by OGC. Similar to GeoJSON, the geometry and properties of the spatial feature are represented.
- **Google Keyhole Markup Language (KML/KMZ)** - A data format developed by Keyhole Inc and later acquired by Google. Later it was taken as a standard of OGC. It is also based on XML grammar and shares some of the data structure as GML.
- **Scalable Vector Graphics (SVG)** - A vector data format developed by the World Wide Web Consortium (W3C) as an open standard. The rendering of SVG is supported by most web browsers.

Table 2: The properties of vector encoding types

Encoding	**File extension**	**Open format or not**	**OGC standardized format or not**
Shapefile	.SHP, .DBF, .SHX	Yes	No
GeoJSON	.GEOJSON .JSON	Yes	No
GML	.GML	Yes	Yes
KML	.KML	Yes	Yes
SVG	.SVG	Yes	No

Raster encoding:

- **TIFF** - A popular data format to store raster images.
- **GeoTIFF** - A standard that involving geo-reference information based on the TIFF format. It is widely used by remote sensing and GIS applications.
- **Joint Photographic Expert Group image (JPEG)** - A commonly used compression method for raster images(*JPEG - Wikipedia*, n.d.).
- **Portable Network Graphics (PNG)** – A commonly used raster data format.

Table 3: The properties of raster encodings

Encoding	**File extension**	**Open format or not**	**OGC standardized format or not**
TIFF	.TIF .TIFF	Yes	No
GeoTIFF	.TIF	Yes	Yes

	.TIFF .OVR		
JPEG	.JPEG .JPG .JFW ...	Yes	No
PNG	.PNG	Yes	No

2.3.2.2. Spatial database and file-based data

It is convenient to use separate files in geo-web-app in some conditions. For example, the GeoJSON file is supported by many web mapping tools to implement interactive visualization. The advantages of using a database are that it can be queried to request minimum information each time to improve the performance, and the update over data is enabled(*Chapter 9 Databases | Introduction to Web Mapping*, n.d.).
The relational database is a type of database to store data in tables, with records in rows and attributes in columns. And there are connections between tables by a foreign key. To store spatial data in the relational database, SQL query can be used to select features with operations such as filtering and ordering. There are some spatial database types or spatial extensions on some DBMS to support spatial data and spatial functions, in which it is possible to perform a spatial query on the data.
PostGIS is a spatial database extender for the PostgreSQL database. The support of geospatial objects and location queries is enabled(*PostGIS — Spatial and Geographic Objects for PostgreSQL*, n.d.). Both vector and raster data can be represented in PostGIS. The geometry of a vector feature can be in a Well-Known Text (WKT) or Well-Known Binary (WKB) format, with the defined reference system. The non-spatial attributes are stored with the geometry attribute in a table. Advanced spatial queries and operations can be done on the spatial features, such as to calculate the distance between two polygons. Raster data can also be loaded into a PostGIS table with some query and operation functions allowed.

2.3.2.3. Geospatial service

The geo resources to be presented in the web browser can be retrieved from a data store directly, or a geospatial server that holding and managing geospatial data and images. The setting up of a geospatial server can be implemented by software like GeoServer. Sometimes the service used by an application is from an external service provider. There are many free and open geospatial service resources in various fields provided by organizations.
There are OGC standards to specify the interface of different kinds of geo services. The OGC web services include map services like Web Map Service (WMS), spatial data services like Web Feature Service (WFS) and geoprocessing services like Web Processing Service (WPS).
To display a map layer on the web browser, a map service can take the responsibility to produce the image and send to the client. Generally there are two kinds of map services commonly used in web mapping: dynamic web map services and static web map services. According to a dynamic web map service, the image is drawn by the server when getting a request from client and then returned to the client. While a static web map service is to send the image pre-drawn on the server to the client. The advantages of the dynamic map service over the static one are the capability to draw pictures with latest data, and the flexibility in setting styles. The disadvantage can be the influence on performance due to the operation of drawing images. The map services can also be grouped into common map services and tiled map services by whether to generate the map in tiles.
The Web Map Service (WMS) defined by OGC specifies the interface that the client getting images drawn by the server. A request to retrieve the image can be sent to the server by a GetMap operation. The parameters to request the image includes the extent of the map, the projection, the layers, etc. And the image returned to the client is in the format of JPG, GIF or PNG. The WMS standard allows applying styles to the layers in a flexible way, which is implemented by the Styled Layer Descriptor (SLD) file.

The Web Map Tile Service (WMTS) standards defined by OGC provide a based solution to serve digital maps using predefined image tiles(Maso et al., 2010). It is a complementation of the WMS. Tiled maps are small square-shaped "chunks" of map pre-drawn and stored in the server. When the user navigates the map, the server returns the tiles to the client. The tiled service is commonly used for a basemap layer. Despite the map services, there are also standards on spatial data service which defines the protocol to transfer vector or raster data from the server to the client. The spatial data service enables the analysis and editing on the features. The OGC Web Feature Service (WFS) standardizes the service to send vector geometries and attributes to the client. The vector feature returned is in the format of GML.
The OGC Web Coverage Service (WCS) standardize the service to transfer raster data in the format like GeoTIFF, and NETCDF.

2.3.3. WebGIS Architecture

These WebGIS technologies make the architecture of geo-web apps different from common web apps, e.g., there is always a geospatial server providing spatial data service or GIS functionality(Agrawal & Gupta, 2017). As WebGIS architecture types, Client-Server Architecture, Service-Oriented Architecture and Spatial Cloud Computing are discussed.

2.3.3.1. Client-Server Architecture

Client-server Architecture follows a traditional web application architecture. Thin client architecture and thick client architecture are different approaches of this architecture type.
Thin client architecture handles most of the processing on the server side. The client only needs to send the HTTP request to server and display the result, which can be image formats such as JPEG and PNG. The render of vector data on client side is always not possible. The disadvantage of this low-cost solution is the limitation of the functionality on client side. Figure 3 displays the components of thin client architecture.

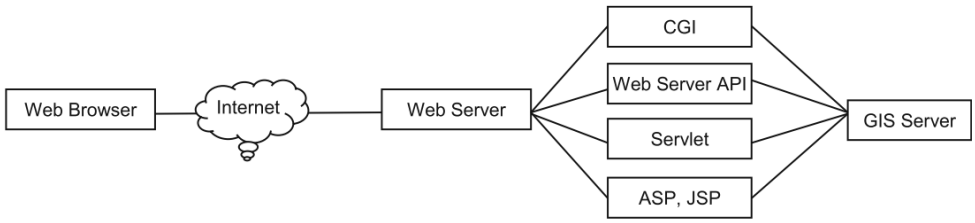


Figure 3: Thin client architecture(Alesheikh & Helai, 2002)

Thick client architecture owns a powerful client which can perform processing as well, with plug-ins, Applet, JavaScript or ActiveX. The client can retrieve raw data from the server and then do manipulations like filtering. This architecture reduces the responsibility of the server and makes the running of the client system in low connection with the server possible. Figure 4 presents the structure of Thick Client Architecture.

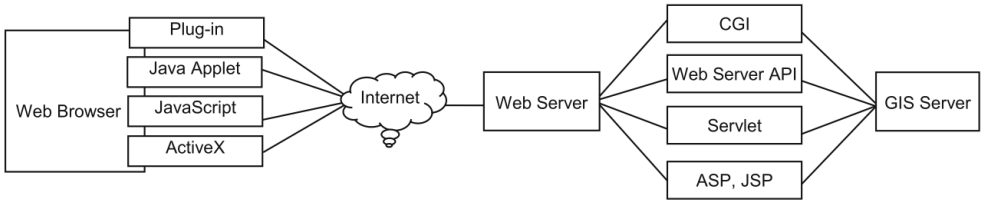


Figure 4: Thick client architecture(Alesheikh & Helai, 2002)

2.3.3.2. Service-Oriented Architecture

Service-Oriented Architecture (SOA) suggests that geospatial web services work as stand-alone solutions, by building a temporary relationship between service provider and consumer(Gu & Van Vliet, 2009). SOA follows the principles of loose coupling, reusability, discoverability and governance. The GIS SOA architecture is presented in Figure 5.

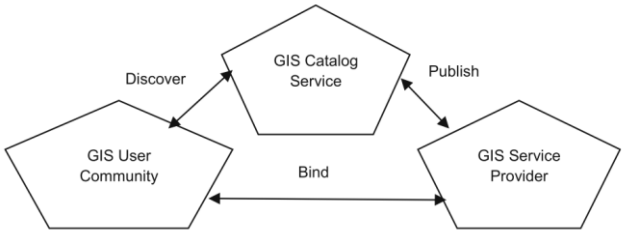


Figure 5: GIS service-oriented architecture(Vaccari et al., 2009)

2.3.3.3. Spatial Cloud Computing

In cloud computing architecture, the facilities are provided over the Internet(Agrawal & Gupta, 2017). Based on this new technology， Behzad et al. proposed a Spatial Cloud Computing Architecture, in which the spatial data processing and visualization can be manipulated in the cloud. Spatial Cloud Computing provides geo resources with high flexibility and low costs.

3. METHODOLOGY

The methodology used in this research is introduced in this chapter. Section 3.1 gives an overview of the research framework, including the methods of each step. From Section 3.2 to 3.4 the major methodology is discussed.

3.1. Research Framework and Methodology Overview

This section gives an overview of the methodology. The objectives to be achieved are identification on common concerns, design of an architectural pattern and an implementation of the pattern. The ongoing of project follows this sequence.

Pattern targets to solve recurrent problems, so the identification of these problems, and the mapping from problem domain to solution domain concern. To reach the common concerns and basic implementation elements, I follow the method proposed by Shalloway and Trott, to used commonality and variability analysis. The commonality analysis requires considering the use case. To better describe the requirements of geo-web apps, a typical application, Toponamenzoeker is analyzed, together with a literature review on concerned questions like map interaction types, and implementations of each concern. As a result of this step, the common concepts with a discussion on implementations are derived.

After finalizing the analysis in problem domain, I design an architectural pattern based on the basic common concepts and implementation analysis. To define the architectural elements, I use the architectural viewtypes of Component-and-Connector and Layered. Elements are structured following these basic structures, and the identification of each element is defined by the requirements, concepts, implementations concluded in the previous step and literature review on some existing patterns. The representation of the architectural pattern follows the format proposed by Gamma et al(Gamma et al., 1995). And the graphic notation uses the Unified Modeling Language(UML).

The last step focuses on the implementation of the proposed pattern. A geo web app, Pandviewer is implemented based on the pattern. The implementation process is modified from the WebGIS development cycle, which is followed by an evaluation of the pattern according to the OO principles and implementation ways.

Figure 6 illustrates the workflow of the research with the methodology used. The knowledge base of the determination of methodology is discussed in Chapter 2. And the major methodology, including the commonality and variability analysis, architectural viewtypes and basic patterns, and implementation process is introduced in the following sections. The other methods, such as the reference application, modeling language and representation of pattern are explained in the corresponding process and result chapters.

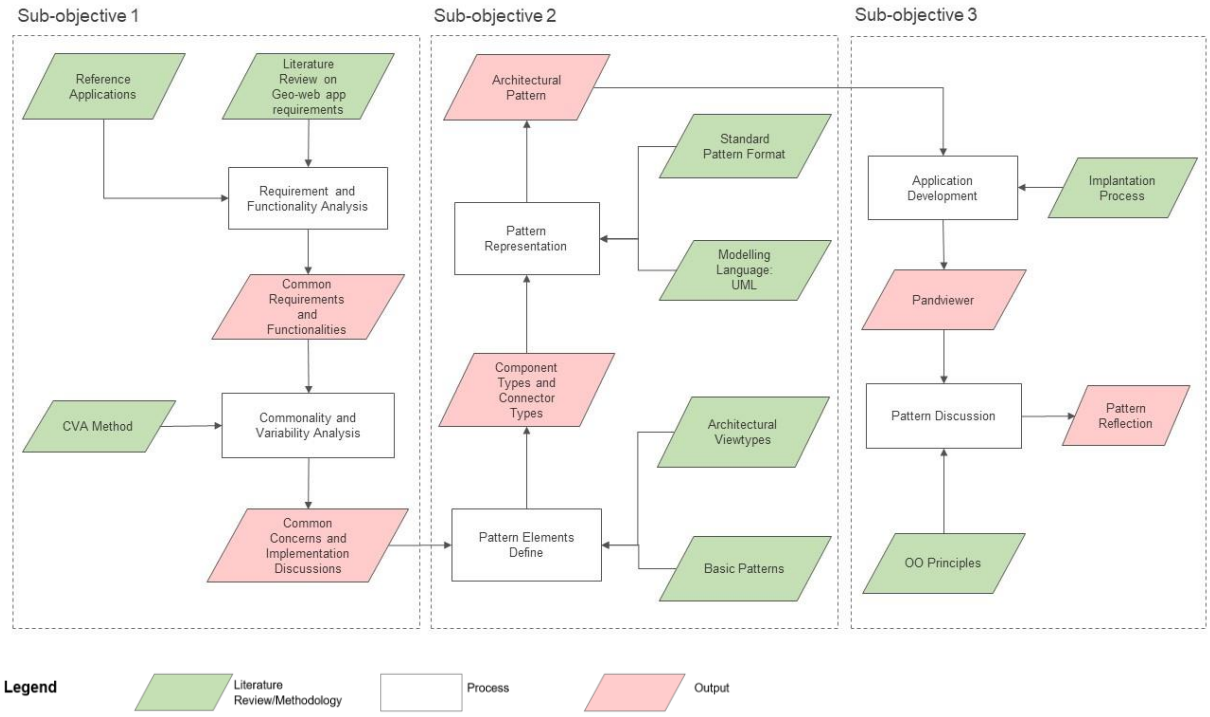


Figure 6: Workflow of the research

3.2. Commonality and Variability Analysis

This section introduces the commonality and variability analysis, as the methodology to conclude the common concepts in problem domain and variabilities in solution domain.

Commonality analysis and variability analysis relates to the conceptual view of the problem domain and implementation respectively(Shalloway & Trott, 2002). Figure 7 illustrates the relationship between commonality/variability analysis, perspective, and abstract classes. As a result, the commonalities define the abstract classes, and the variations within each commonality define derivations of abstract classes.

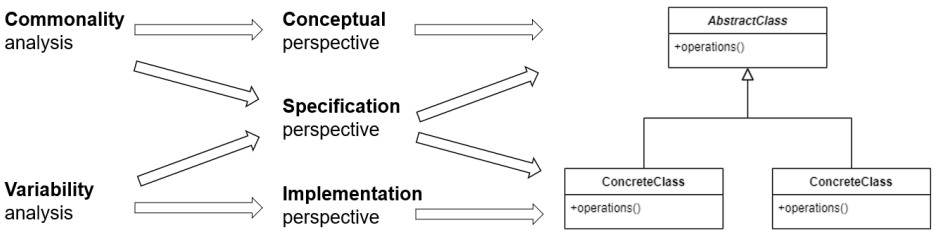


Figure 7: Relationship between commonality/variability analysis, perspective, and abstract classes(Shalloway & Trott, 2002)

Within an architecture, commonality reveals the structure that is unlikely to change, while variability reflects the structure that is likely to change, and depends on the specific commonality. It provides a strategy to form a conceptual design with OO methodology.

Shalloway et al. present an approach to use commonality and variability analysis to derive a design pattern, with an example of the Bridge pattern(Shalloway & Trott, 2002). It starts from the definition of requirements, to the determination of common concepts and variations, and to the illustration of the pattern.

This research follows the same strategy to analyze the problem domain. The user requirements and functionality of geo-web apps are enumerated, to reach the common concerns and implementation variations. These concepts with discussion on implementations are used as basic elements for the architectural pattern.

3.3. Architectural Viewtypes and Basic Patterns

This section introduces the architectural view types used for sub-objective2, as the method to construct pattern elements. The architectural views can be classified by a large granularity into structural view, behavioural view,. etc or a small granularity into pipe-and-filter view or a client-server view, etc. To design the architectural pattern I follow the middle path between these two approaches proposed by Avgeriou et al(Avgeriou & Zdun, 2005). The Component-and-Connector view is taken as the basic scheme to construct the elements and their relations. And in a specific perspective, the Layered View is taken as the primary view type, to decompose the system into several interactive parts.

3.3.1. Component-and-Connector View

Clements et al. indicate that Component and-Connector view concerns the elements with run-time presence(Clements et al., 2003). Component and connector types, constraints for allowed relations, properties of components and connectors and topologic constraints are elements of a general Component-and-Connector viewtype(Béjar et al., 2009). This viewtype provides a standard method to describe the elements of an architectural pattern/style. In my design I give a discussion and explanation on the component types, connector types and constraints on relations, to introduce how these elements are derived and constitute a whole architectural pattern.

3.3.2. Layered view

The Layered view is to decompose the system into several interactive parts, which can separate the concerns. There are elements with different responsibilities in the architecture of a geo-web app, and the elements can be well structured by divided into layers. As an instance of the Layered View, the Layers is an architectural pattern in which high-level layers depend on low-level layers to perform their functionality. Figure 8 presents the structure of the Layers architectural pattern.
In the previous steps, the concepts to be included in the architecture are presented. To structure the architecture a Layered view can decouple the responsibilities according to their levels. Some concepts can be collected with a layer and some concepts can be divided into different layers according to the separation of responsibilities. The Layered view enables a clear structure of architecture.

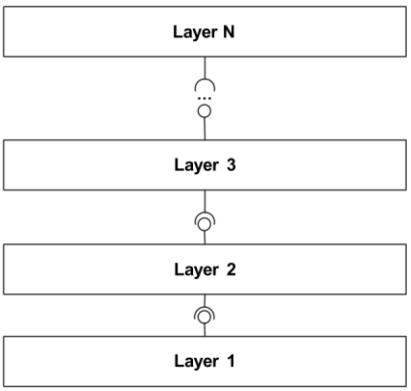


Figure 8: Structure of the Layers pattern(Avgeriou & Zdun, 2005).

3.3.3. Three-Layer architectural pattern

The three-layer architectural pattern separates a system into three logic components: presentation layer, business layer, and data layer. The presentation layer handles the user interaction with the application. The business layers deal with the core functionality and the data layer provides access to data(*Application Architecture Guide - Chapter 9 - Layers and Tiers - Guidance Share*, n.d.). The structure of this architectural pattern is illustrated in Figure 9. As a generic architectural pattern, three-layer architecture is used as the basis of my design.

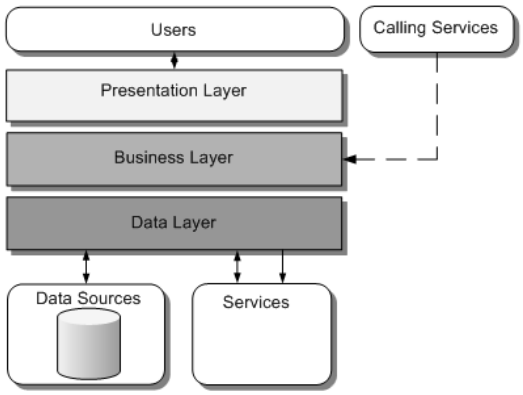


Figure 9: Structure of the three-layer architectural pattern(*Application Architecture Guide - Chapter 9 - Layers and Tiers - Guidance Share*, n.d.)

3.3.4. Client-server architectural style

Client-server architectural style defines a mechanism that client and server, as two components, communicate with each other in that the client request for data and the server provides it. Figure 10 presents an example of the client-server architecture with multiple clients.
This architectural style provides a definition of component types and communication ways for web applications. This strategy is used in my design to define the client-server relation.

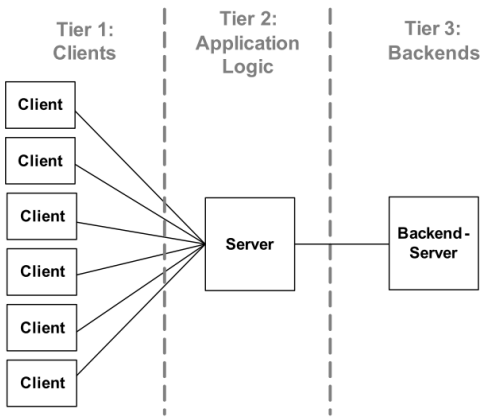


Figure 10: Three-tier client-server architecture(Avgeriou & Zdun, 2005)

3.3.5. Observer pattern

Observer is a design pattern that defines a publish-subscribe relationship between objects. It describes a one-to-many dependency that, the changes on one object notifies its observers to update states.
The structure of Observer pattern is illustrated by a class diagram in Figure 11. This pattern is used as a relation type between presentation components and interaction components of the architectural pattern.

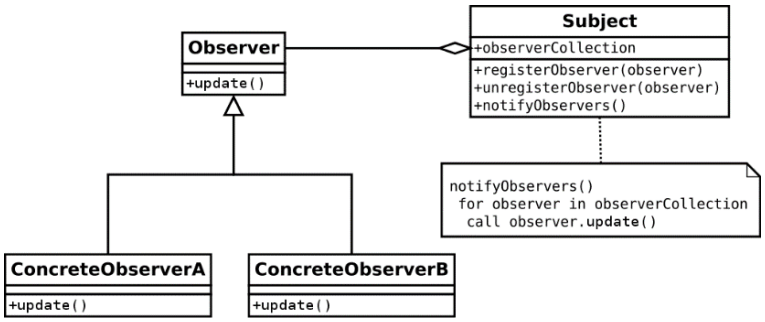


Figure 11: Class diagram of Observer pattern(*Observer Pattern - Wikiwand*, n.d.)

3.4. Implementation Process

This section discusses the process to implement a geo-web app, by explaining the workflow, and the task of each step.

The WebGIS developing cycle includes 8 activities from user requirement analysis to WebGIS ongoing use and maintenance(Luqun et al., 2002). Figure 12 presents the activities of the WebGIS developing cycle.

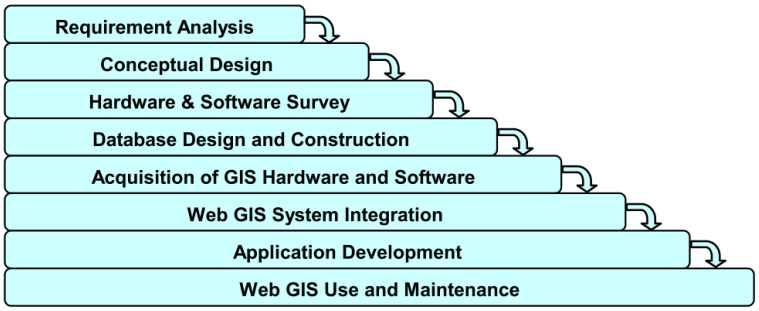


Figure 12: WebGIS developing cycle(Luqun et al., 2002)

According to implementing a geo-web app with the architectural pattern, an implementation workflow can be applied based on the developing cycle with the following activities:

- **Requirement Analysis** - Analyze the target user and purpose of the application, and define the required functionalities.
- **Data Collection** - Define the source of spatial data. The determination of data store types makes sense to the architecture design.
- **Architecture Design** - Based on the functionalities, the architecture of the application can be designed by applying the components and connectors of the pattern.
- **Techniques Survey** - Select the techniques to build the application.
- **Application Development** - Develop the application following the architecture.

The modified implementation process is presented in Figure 13.

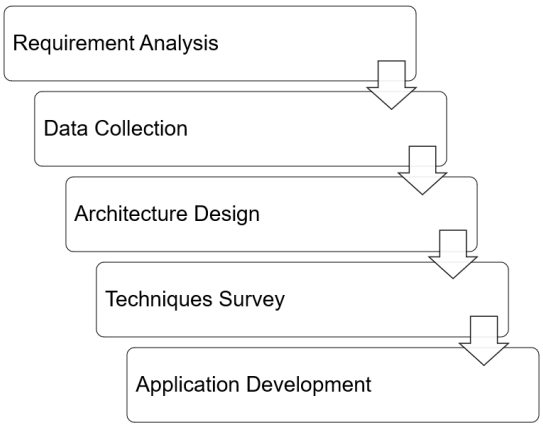


Figure 13: Implementation process of the research

3.4.1. Requirement Analysis

The requirement analysis includes an overview of the application intent and a conclusion on the functionalities. The following questions should be answered in this activity:

- What is the purpose of the application?
- What is the target user group of the application?
- What are the requirements of map display? Which map elements should be involved? What are the map layers to be included?
- What are the requirements of component interaction?
- What is the user interface design of the application?

3.4.2. Data Collection

The establishment of a geo-web app requires a data store, which is the Data Repository Layer in the architecture. This process contains collecting data sources and designing data store types. The relevant questions are as follows:

- What is the data source of the application?
- What is the type of spatial data, is it raster or vector?
- What are the data repository types of the application?

3.4.3. Architecture Design

The architecture design of the application makes use of the architectural pattern, which provides a template for the design. The components and connectors can be defined according to the pattern. Concerning questions are as follows:

- Which components in the pattern are required for the application and which components are not necessary?
- What are the concrete responsibilities of the components in architecture?

3.4.4. Techniques Survey

In this process, the techniques to build a geo-web app are selected. There are questions according to the relevant techniques:

- What are the data storage mechanism, e.g., database management system types and geospatial service types of the application?
- What are the programming languages of the application?
- What are the frameworks for back-end and front-end respectively?

- What are the spatial data processing libraries in the back-end?
- What are the web mapping libraries in the front-end?

3.4.5. Application Development

After the definition of architecture and techniques, the application can be developed. This activity includes the deployment of data and service and coding.

- How to map from the logic architecture to physical architecture?
- How to implement the architectural elements with the techniques?

4. COMMON CONCERNS IDENTIFICATION

In this chapter, we explore the common concerns in developing a geo-web app with rich interactions. The Toponamenzoeker (Placename Finder) app(*Toponamenzoeker*, n.d.), a typical single-page web application with web mapping functionality is used as an example. In section 4.2 we start to explore the user requirements and functionalities of geo-web apps. In Section 4.2 and 4.3, the commonality and variability are discussed respectively. Finally, in section 4.4 a summary of this step is given.

4.1. User Requirements and Functionality

The user groups of geo-web apps can vary. For example, anyone interested in a particular spatial question with themes like weather or traffic can be the target user group of a geo-web app. In general terms, the intent of a geo-web app is to allow one to analyze spatial data and visualize the results. To that end, a geo-web app provides some functionality such as searching and filtering to let the user explore the data. Google Map is a geo-web app for geospatial data visualizations(*Google Maps - Wikipedia*, n.d.). Besides the requirements of displaying geo-objects, there are functionalities of directions and transit, traffic conditions, .etc. Users can view the map with features and find locations by Google Map. Figure 14 gives an example that the user searches for the ITC building in Google Map.

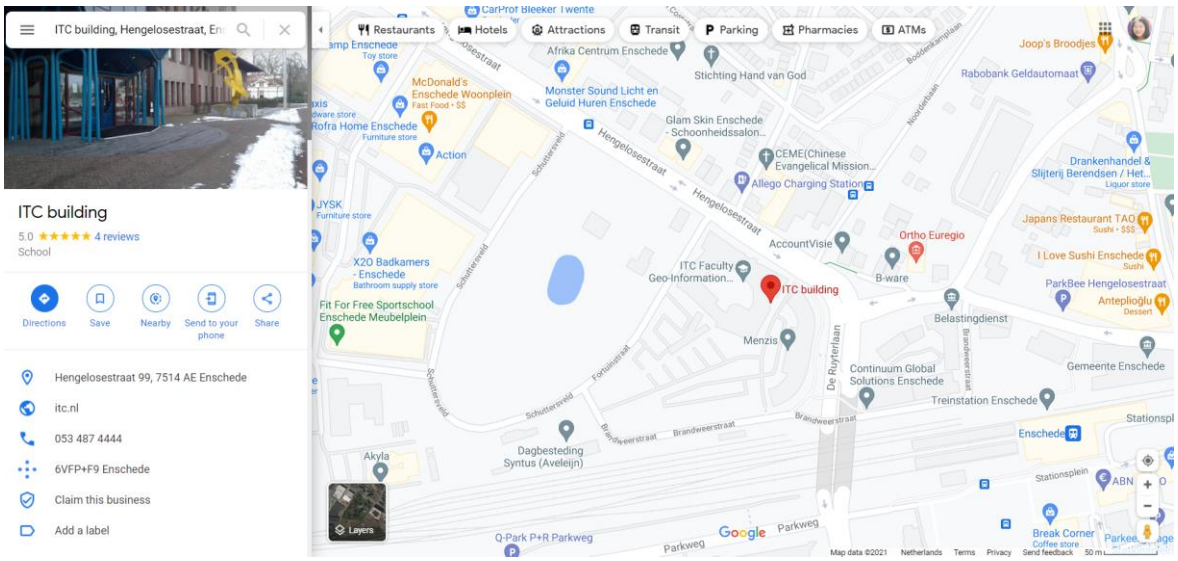


Figure 14: Searching for the ITC building in Google Map(*ITC Building - Google Maps*, n.d.)

The Covid-19 Dashboard is a geo-web app developed by Johns Hopkins Coronavirus Resource Center (CRC) for real-time data visualization(*About Us - Johns Hopkins Coronavirus Resource Center*, n.d.). The statistics of cases and deaths are presented in the map, chart and list. Users are able to select a region on the map or search a region for the statistics. The user interface of this dashboard is presented in Figure 15.

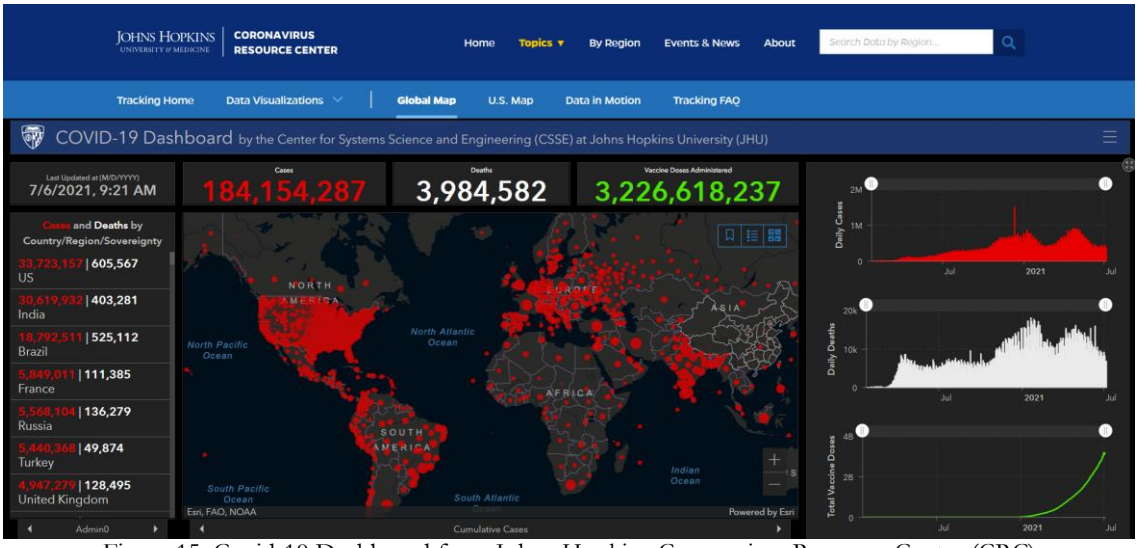


Figure 15: Covid-19 Dashboard from Johns Hopkins Coronavirus Resource Center (CRC)

Toponamenzoeker(Placename Finder) is a geo-web app developed by Kadaster, to provide a visualization of the land objects in the Netherlands(*Toponamenzoeker*, n.d.). Users are able to search for objects and select objects for information. Figure 16 illustrates that the user searches for 'Enschede' in Toponamenzoeker and gets results.

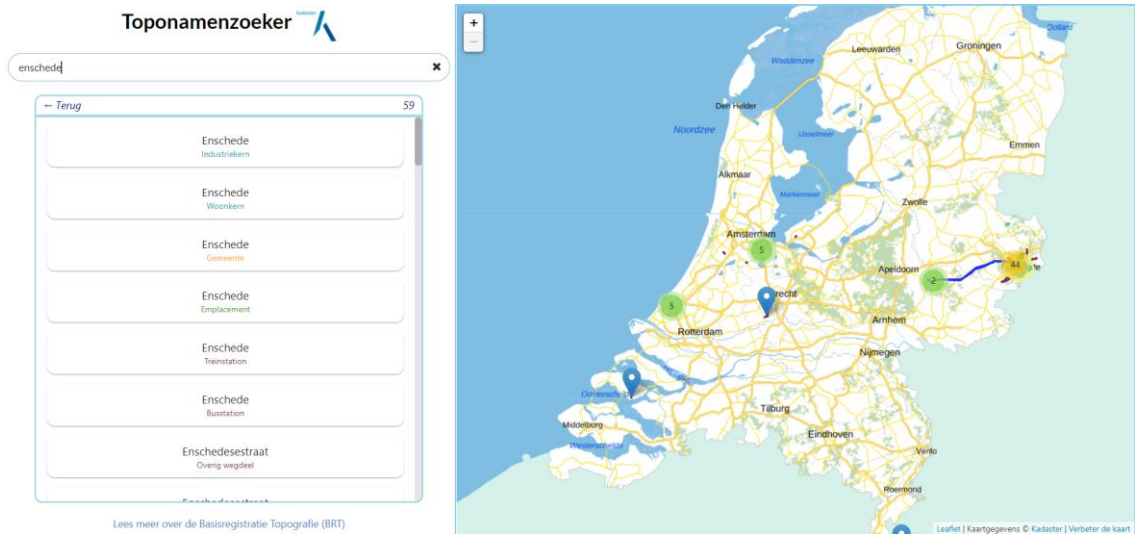


Figure 16: Searching for 'Enschede' in Toponamenzoeker(*Toponamenzoeker*, n.d.)

These geo-web apps are applied to different fields, to serve for geodata visualization. And the requirements to visualize geo features interactively, search for geo objects and select geo objects are always included. To analyze the common requirements and functionalities of geo-web apps, we use Toponamenzoeker, which is a typical application as a reference.

The target user group of Toponamenzoeker is non-expert users interested in information about land objects registered in the base topographic register of the Netherlands BRT (Basisregistratie Topografie). BRT is a dataset produced by the Dutch Kadaster with registrations of land objects such as roads, water and buildings in the Netherlands. This application provides functions to locate and visualize the objects available in the BRT. The user should be able to search for the objects by a location on the map, or by a name on a search bar. The shape and attributes of the resulting objects are then presented to the user on various page containers. Furthermore, the user is able to select a particular object to view specific details.

The user requirements of Toponamenzoeker are presented in Figure 17 and explained in Table 4. The use cases enable the identification of the interactions between users and the app.

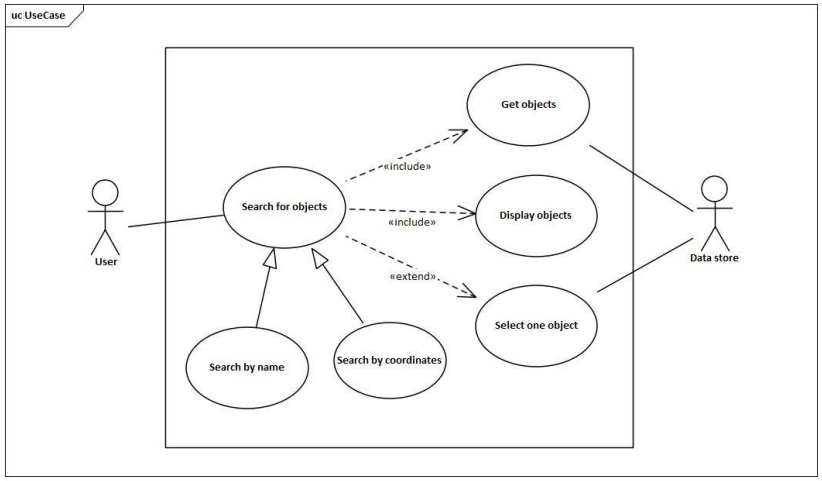


Figure 17: Use case diagram of Toponamenzoeker

Table 4: Description of use cases of Toponamenzoeker

Use case	**Description**
Search for objects	The user can search for land objects using conditions.
Search by name	The user can search for land objects by their name. The search function is triggered by just typing in the search bar.
Search by coordinates	The user can search for land objects nearby a given location. The user can right-click on the map to start the search.
Get objects	Objects' data is retrieved from the data store as a response to the search interaction.
Display objects	Data of the identified objects is displayed on the map and the object list container. Features are drawn in an interactive layer, and the name and type of the features are displayed as a list on the left bar.
Select one object	From the matching group of objects the user can select one object by clicking on the item on the list or on the map. And then the interactive layer is refreshed to display only the selected object. Also, the left bar is updated to display the details of that object replacing the existing list.

The typical functionality of geo-web apps includes map display and component interaction. The map display is the capability to present a web map with spatial information displayed. And component interaction to let the user explore the spatial information by interacting with the map or other elements such as tables, graphs, etc. The interaction can be categorized by their purposes into map navigation, searching and filtering, information retrieval and data manipulation (Tolochko, 2016).

4.1.1. Map display

Map Display is the basic functionality of a geo-web app, to display the spatial information as a web map., which starts with the layer display. The OpenGIS model of portrayal workflow implements interactive portrayal (Luqun et al., 2002), as is presented in Figure 18. The expression of layer display can be based on the OpenGIS model, to be divided into four steps: Data Selection, Visualization Element Generator, Render and Display.
The Data Selection process is used to retrieve data from a geospatial data store based on query constraints. And the Visualization Element Generator converts the retrieved data into display elements by adding styles. The Render is to generate the image from spatial features. And the Display presents the map layer to the users(Luqun et al., 2002).

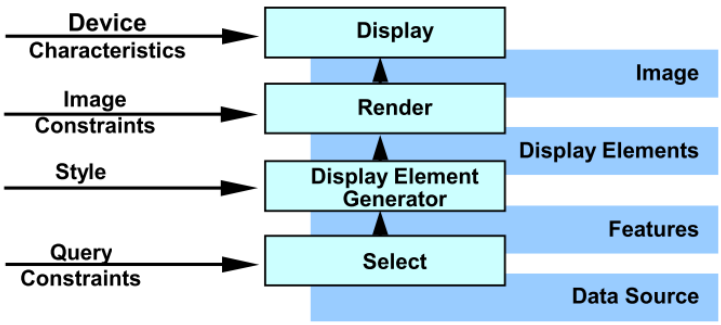


Figure 18: OpenGIS model of portrayal workflow(Luqun et al., 2002)

In addition to the layer display, the web mapping also involves the management of layers, e.g., the overlay ordering of layers, the view settings of the map, e.g., the defining of projection and extent and the generating and displaying of additional map components.
In the case of Toponamenzoeker, the requirements of map display functionality are presented in Table 5. The map contains three layers, the basemap layer, the feature layer and the cluster feature layer. The basemap layer is a static layer with pre-drawn images, and the feature layer and cluster feature layers are interactive layers that follows the layer display workflow discussed.

Table 5: Map display functionalities of Toponamenzoeker

Functionality	**Description**
Map layer display (basemap layer)	A standard background map of the Netherlands.
Map layer display (feature layer)	The feature layer with selected spatial features with symbolizing based on feature attributes (types).
Map layer display (cluster feature layer)	The cluster feature layer with selected spatial features is visible when the map is on a large scale.
Layer organizing	The basemap layer should be at the bottom, and the visibility of the feature layer and cluster feature layer should be controlled by the zoom level.
Map view setting	The initial map should be set to the scale that fits the whole country.
Map components display	A zoom control to modify the visual scale, and the attribution text to acknowledge the data sources.

4.1.2. Map navigation

The map navigation interactions are meant to support the functions of panning and zooming of the map. This is a necessary function to let the user change the scale of the map and navigate to relevant locations. This functionality can be implemented as buttons or using scrolling to zoom and dragging to pan. The map navigation interaction enables the user-friendly visualization of a web map.
Table 6 presents the map navigation interactions of Toponamenzoeker. In addition to the basic map zoom control, there is a functionality to zoom to the interested area when the user selects an object.

Table 6: Map navigation interactions of Toponamenzoeker

Map navigation interaction	**Description**
Scroll to zoom	The user can use the scroll wheel to zoom in or zoom out of the map.
Drag to pan	The user can drag the map to pan.
Zoom by a control button	The user can zoom in or zoom out by a control button.
Zoom and pan by object selection	When the user selects an object, the map will be panned to that area, and the scale will be changed to fit the visualization of the object.
Reset	The user can click the header to reset the scale of the map.

4.1.3. Search and filter

Searching and filtering are essential features of a geo-web app. The user can find the features by keywords or phrases which are used as conditions for searching and filtering data from the source. It makes it possible for users to explore spatial data in different ways. Searching and filtering can be used to answer questions of 'where', 'when' and 'what', which are to find objects based on location, time and name or descriptions respectively.

- The search operation finds objects based on user input, which can be in the form of typing in the text box or clicking on map. The user input is used to set query constraints and retrieve matching results. Search conditions in one feature collection, are applied to the features' attributes or geometries. Search conditions across multiple feature collections are applied to the relationships between features, e.g., the intersect relation.
- The filter operation is to set constraints on the feature display. It enables the selection of spatial features by conditions. Also the selection by numeric values is possible. The distinction between search and filter is that search starts from nothing, but filter begins with a list of results.

The search and filter functionality work together with the map display functionality in that it always triggers a process of rendering map layers, from Data Selection to Display.
The search and filter interaction is the core functionality of Toponamenzoeker. Search options are described in the use case diagram in Figure 17, also in Table 7. The Search functionality is to a get set of features, and the Select object functionality is to get a unique feature. The Select object functionality can be taken as a combination of Filter and Information Retrieval.

Table 7: Search and filter interactions of Toponamenzoeker

Search or filter functions	**Description**
Coordinate search	The user can trigger a search by right-clicking on a location on the map.

Text search	The user can trigger a search by type the name in the search box.
Select by choosing from search results	The user can select an object from the candidate search results in the list on the left bar.
Select by clicking spatial feature	The user can select an object from the displayed features on the map.

4.1.4. Information retrieval

Information retrieval is a kind of interaction that lets the user get more detailed information on an object than that displayed on the map. It can be in the form of a floating tooltip or popup, or a fixed panel. Usually this kind of interaction is commonly used in vector layers, to enable displaying additional information of a spatial feature on mouse hover or clicking.
The information retrieval in Toponamenzoeker can be in the pop-up of a feature, in the side panel, or hyperlink to the data source. The side-panel information display works together with the Select interaction. The information retrieval interactions in Toponamenzoeker are described in Table 8.

Table 8: Information retrieval interactions of Toponamenzoeker

Information retrieval interactions	**Description**
Feature pop-up	The user can view the name and type of an object in pop-up by moving the mouse over it.
Information in the side panel	The user can view details of an object in the side panel by selecting the object.
Hyperlink to the data source	The user can view the source information of an object with the persistent identifier (URI) of the object, by clicking on the hyperlink displayed in the side panel.

4.1.5. Map configuration

The map configuration interaction allows the operation to change or add content to the layers. There are functionalities to add or remove overlays, switch geographic units and customize map symbology in the application. The users can interact with controllers to implement these functionalities, which is Map configuration interaction. The map configuration allows changing the appearance of the map with the current data or retrieved data.
Toponamenzoeker includes the map configuration interaction to switch between the feature layer and the cluster feature layer. When the zoom of the map is larger than a level, the feature layer will replace the cluster feature layer. Table 9 presents the map configuration interactions in this application.

Table 9: Map configuration interactions of Toponamenzoeker

Information retrieval interactions	**Description**
Switching feature layers	The feature layer appears on a large scale and the cluster layer appears on a small scale.

The user requirements and functionalities of geo-web apps are discussed above. The derived common functionalities are not exhaustive. There can be other requirements not listed. In the next section, the common concerns can be analyzed from these discussed requirements.

4.2. Commonality Analysis

Commonality analysis is to find concepts that are unchanged in a system under the different user requirements(Shalloway & Trott, 2002). The common concepts explain the concerns in problem domain and can serve as abstract components of the system architecture. According to object-oriented design, commonality analysis help define abstract classes. In this case, we use commonality analysis to decompose the problem domain into responsibilities.

The functionalities discussed in 4.2 can be divided or combined into common concerns. The Map display functionality involves the construction of layers and other map elements. The Map navigation functionality, which is to modify the view of the map, can be combined with the Map display into a MapManager concept. The creation of layers can be extracted as an individual concept. There are steps of Data Selection, Visualization Element Generator, Render and Display according to the creation of layers. The Data Selection and Visualization Element Generator can be taken as separate concerns of DataRetrievel and FeatureStyling. And a DataProcessor concept can be added to manipulate spatial data for use. The interactions of Search and filter, Information retrieval and Map configuration can be combined to an Interactor concept, which accepts user event and has an influence on components.

Based on the required functionalities of a geo-web app discussed in 4.2, the following common concepts can be extracted, with their responsibilities.

- **MapManager** - The MapManager concept concerns the organizing and displaying of a map, including the construct of map elements, e.g., layers and legend. Also, it concerns the arrangement of the map view, to enable the Map Navigation interaction by defining operations.
- **LayerCreator** - The LayerCreator concept concerns the creation and rendering of a map layer, which is a component of a map. The steps of Data Selection, Visualization Element Generator and Render are organized by the concept.
- **FeatureStyling** - The FeatureStyling concept concerns the step of Visualization Element Generator in the layer display workflow, to assign styles to spatial data.
- **Interactor** - The Interactor concept servers as a mediator component to manipulate the interaction on the map and other components. The logic of the collaboration and interaction methods can be defined by an Interactor.
- **DataRetrieval** - The DataRetrival is responsible for accessing the spatial data for display, which is the Data Selection step. And some interactions also rely on the DataRetrieval concept.
- **DataProcessor** - The DataProcessor manipulates spatial data. In some cases of Search and Filter interaction, the processing on spatial data should be included.

4.3. Variability Analysis

While commonality describes the problem in the conceptual perspective and specification perspective, variability analysis focuses on the specification and implementation perspectives (Shalloway & Trott, 2002). Variability analysis is to find the variations in terms of a given commonality. According to object-oriented design, variability analysis can help define concrete classes.

In section 4.2 we identified the common concerns of a geo-web app system. In section we discuss the variations for each concept. The cases and concrete implementations of Toponamenzoeker in terms of these concepts are discussed, as examples of the variations.

4.3.1. MapManager

The variations on MapManager are not obvious. There are few differences according to the implementations to construct a map within a system design and between several systems.

In the case of Toponamenzoeker, the MapManager is implemented with the functions to set the view, adding the layers and bind the right-click searching interaction. Also the Map Navigation interaction relevant with zoom and pan is defined by MapManager.

4.3.2. LayerCreator

The variations on LayerCreator depend on the type of map layers, since the type influences the way to create a layer. Based on the data source the layer can be categorized into image layer and vector layer. The image layer uses pre-drawn images, or images drawn on the fly by a service. The vector layer uses vector data to render a layer step by step. As a result, the implementation of the layer display workflow varies. For example, The Visualization Element Generator of a vector layer can be implemented by the FeatureStyling class, while that of an image layer can be delegated to another component, according to the type of the image store or service.
In the case of Toponamenzoeker, the basemap layer is an image layer, with the source from a WMTS. The function to generate the layer is simple. The feature layer and cluster feature layer are vector layers. There are functions to retrieve the feature data, assign styles and render the layer.

4.3.3. FeatureStyling

The variations on FeatureStyling can be based on the geometry types of the feature, because there are different methods to symbolize the polygon, polyline, point, .etc. The concrete functions to define styles depend on the use cases. The visual variables can be assigned to the feature for qualitative or quantitative visualization based on attributes.
In terms of Toponamenzoeker, FeatureStyling is required in the symbolization of the feature layer and cluster feature layer. The cluster feature layer is composed of point features, so markers in a circle shape are used to represent features. The feature layer is composed of point and polygon features. For the point feature a marker is used for visualization and for the polygon feature the style of shape is defined, with the visual variable of color based on the 'type' attribute, and a marker is added in the center of the feature.

4.3.4. Interactor

The variations on the Interactor concept depend on the type of interactions and the type of components the interactions are on. Search and Filter, Information retrieval and Map configuration interaction are to generate information from the user input, and take action on the results. There are variations in implementing the process, that whether the application gets the information from the server or from the data already in the client.
In terms of Toponamenzoeker, relevant interactions include searching by coordinates, searching by text, selecting by objects list, selecting by features on the map, feature pop-up and switching between cluster feature layer and feature layer. The implementation of searching and selecting needs to retrieve data from the server, while that of feature pop-up and switching layers uses data that is already in the client.

4.3.5. DataRetrieval

The variations on the DataRetrieval depend on the type of data store. The methods to get data from files or databases vary. The relational database allows queries on the records, which can implement selection on features. There are spatial databases that support spatial query such as intersect to enable the selection based on spatial relations. In terms of retrieving file-based data, the query on a single file is not possible. Also, the file type concerns, to let the application read the file.
According to Toponamenzoeker, the data retrieval process is implemented by SPARQL query and Elasticsearch on the linked datasets. The search and select functionality require the retrieval step, to execute the query with user input as conditions, and return the spatial data.

4.3.6. DataProcessor

The DataProcessor is to deal with the calculation and manipulation of spatial data according to the user requirements. There are variations on that whether the processing is on raw data or on retrieved results, which makes a difference to the implementation. The processing on raw data is implemented on server

side and that is on retrieved results can be implemented on client-side. The design choices depend on the processing types and the requirements on performance.
In terms of Toponamenzoeker, there are methods to manipulate the feature data after retrieving it from the data store, such as to filter the invalid records and transform the encoding of geometry from WKT to GeoJSON. Spatial processing and calculation are not included.

4.4. Summary

With the commonality and variability analysis, the problem domain of a geo-web app is described and the common concepts and variations are analyzed. The concepts can be used as components of the conceptual design, however, there are limitations on this model as a pattern:

1. The separation of responsibility into concepts is coarse-grained. There can still be decoupling on the concepts to make an understandable and practical architectural pattern.
2. The collaboration between components is not well analyzed and defined.
3. The mechanism to manage the data flow from concepts is not defined.
4. The feature of web applications is not reflected in the design.

To solve these problems, we will improve the conceptual design from the perspective of architectural views. The commonality and variability analysis define the basic structure of a geo-web app system, and in Chapter 5 an architectural pattern will be retrieved and presented as a reusable solution.

5. A PATTERN FOR GEO-WEB APP

In this chapter an architectural pattern is proposed for the design of a geo web application, by defining the pattern elements and then representing the architectural pattern. The components and connectors of the architectural pattern are analyzed in 5.1 according to the common concerns discussed in Chapter 4. And in 5.2 a formal representation of the architectural pattern is given, from Intent to Related Patterns.

5.1. Component and Connector Types

In this section, the components and connectors of the architectural pattern are defined, under the selected architectural viewtypes: Component-and-Connectors and Layered. The concepts with their variations proposed in Chapter 4 are used as basic components. And the determination of element types is based on the common concepts, variations and basic patterns.

5.1.1. Component types

The components in this pattern are the specifications of the layers and the internal elements of each layer. The layers are defined by the different levels of responsibilities of the concepts, which are followed by the required components in each layer.

5.1.1.1. Layer types

The layer types are defined following the rule of Layers pattern that high-level layers depend on low-level layers. The system can then be decoupled in a vertical manner. The three-layer architecture style divides the system into a presentation layer, a business layer and a data layer. The layers can be sub-divided and modified to adapt to the case of the geo-web app. From the requirements analysis in Chapter 4, we can make a summary about adapting the layers architecture to geo-web apps:

- The business layer of the system, can be weakened because the emphasis of the system is not on the business process but on the visualization and user interactions to update the visualization.
- The presentation layer, which manages the view and user interactions, is the emphasis of the system, and the layer can be sub-divided into specific layers to perform different tasks.

Based on these conclusions, the concepts can be assigned to the following layers:

- **Data Repository Layer** - It is the lowest level layer of the system, to hold the spatial and non-spatial data store of the application.
- **Data Access Layer** – It corresponds to the DataRetrival and DataProcessor concept, to process the data and retrieve the data on server side. This layer relies on the Data Repository Layer.
- **Communicator Layer** – It is defined as a layer in client side to communicate with Data Access Layer, and the processing task after retrieving is managed by this layer.
- **Interactor layer** - It takes the responsibility of managing the process flow and state management logic of interactions, which is in line with the Interactor concept.
- **Presentation layer** - It manages the user interface and receives user inputs. The concepts of MapManager, LayerCreator, FeatureStyling are covered by this layer.

Figure 19 presents a division of the layer components for the pattern.

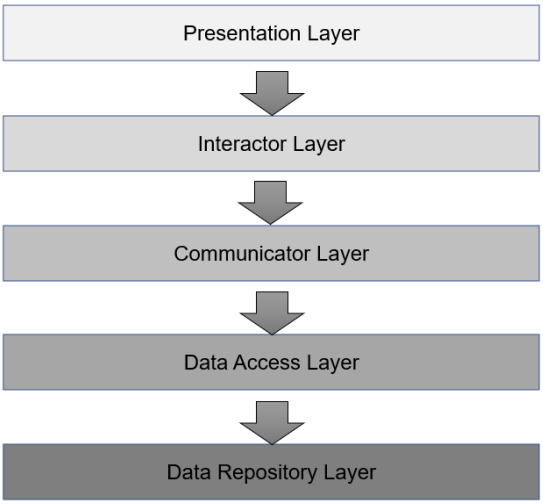


Figure 19: Structure of layer components of the architectural pattern

5.1.1.2. Data Repository Layer

The data repository holds collections of data required for the application. According to spatial data, raster and vector data can be stored in files or in databases. And there can be geospatial services providing spatial data or images. Sometimes there is a need to deploy non-spatial data for use. The techniques to store spatial data are discussed in sub-section 2.3.2. This Layer is composed of these data repositories, with mechanisms to store and maintain the data.

5.1.1.3. Components in Data Access Layer

The concerns of a Data Access Layer include the data access approach, the mapping from data structure to data source, and the way to connect to the data source(*Application Architecture Guide - Chapter 12 - Data Access Layer Guidelines - Guidance Share*, n.d.). In terms of retrieving data from a data store, there can be a Data Retrieval component with functions to connect to the data store and access data. In terms of retrieving data from a geospatial service, a Service Agent component can be introduced to manage the semantics of communicating with the service. The variations in DataRetrieval concept are reflected in these two components. And the Data Processor component, which is in line with the DataProcessor concept can be used to manage data manipulation, such as geo-processing. The description of the components is presented in Table 10.

Table 10: Components in Data Access Layer of the architectural pattern

Component	**Responsibility**
Data Retrieval	Connect to data repository and access data
Service Agent	Communicate with geospatial services and access data
Data Processor	Manipulate data

5.1.1.4. Components in Communicator Layer

The Communicator Layer serves as a mediator between Data Access Layer and Interactor Layer. The separation of Data Access and Communicator implements the client-server pattern. The Data Access layer is on the server-side and the Communicator layer is on the client-side. They communicate with each other with the request-response method to transfer data. The Communicator Layer is responsible for sending requests to get data and treat the results. A Data Communicator component controls the actions of requesting and receiving data, and a Helper component provides methods to perform post-processing on

the data, such as filtering and format conversion. The Data Processor component and Data Helper component implements the variations of DataProcessor concept. The components in Communicator Layer are described in Table 11.

Table 11: Components in Communicator Layer of the architectural pattern

Component	Responsibility
Data Communicator	Send requests to Data Access and get results
Data Helper	Manipulate the results

5.1.1.5. Components in Interactor Layer

The Interactor Layer handles the logic of user interactions. The mechanism to implement the Interactor is not analyzed in Chapter 4. In this sub-section the concrete approach is discussed, and then the required components are retrieved.
The concern of the Interactor Layer is the data flow between Presentation Layer and Communicator Layer. The Interactor captures the user events on the UI components, and then retrieves data from the Communicator, and updates the UI components with the retrieved results. To manage the interaction process, state management is required to control the data flow.
The design takes the scheme of Flux and Redux, by extracting the abstraction part from these mechanisms. Flux and Redux are state managements based on the MVC pattern, but follows a unidirectional data flow. The components of the state manage process are View, Action and Store. The View is the Presentation Layer. The Action is to handle the events triggered by users on the view and process the workflow. The state of the application, which is a set of data, is managed by the State Store. The Action interacts with Communicator to get data, and notify Store to update the state, and then has an impact on the View. The components in Interactor Layer are described in Table 12.

Table 12: Components in Interactor Layer of the architectural pattern

Component	Responsibility
Action	Translate the user events into actions, process the flow by interacting with Communicator Layer, and dispatch the action.
State Store	Hold and manage the state. Notify the Presentation Layer with the change on data to update the view.

5.1.1.6. Components in Presentation Layer

The Presentation Layer manages the UI components and some processing methods on them. Compared with the presentation layer in Three-layer architecture, the focus is on render and display. It concerns the structure and style of UI components, the approach to acquire user input, and a part of user interaction process. The interactions without influencing the application state and retrieving data, are proceeded in the Presentation Layer. In terms of complex interaction the task is delegated to the Interactor Layer. By this method the variations on Interactor concept are reflected. Other concepts relevant to this layer are MapManager and LayerCreator, which controls the structure of map-relevant UI components, and FeatureStyling, which controls the styles. The major components in Presentation Layer are UI Manager, to create and display the whole UI elements of the application.
According to a geo-web app the Map Manager, Layer Creator and Feature Styling methods can be taken as components that help manipulate UI components. Within these components, there are variations in implementations. The components in Presentation Layer are described in Table 13.

Table 13: Components in Presentation Layer of the architectural pattern

Component	Responsibility
UI Manager	Construct and render the component in the client, receive user input, handle user interactions and communicate with Interactor Layer.
Map Manager	Manage the display of the map element.
Layer Creator	Manage the generation of the layer element.
Feature Styling	Manage the symbolizing of features.

5.1.2. Connector types

In this sub-section the connector types are defined, indicating how the components interact with each other. The determination of connector types is based on the architectural view types and some existing architectural patterns/styles. Firstly there is a discussion about connector types between layers, and then connector types between components within each layer are defined.

5.1.2.1. Connectors between Layers

According to the Layered View, the interaction mechanisms between the components are implemented through connectors that include appropriate interfaces, states, and interaction protocols(Avgeriou & Zdun, 2005). In this design, the relation between layers follows the basic scheme that high-level layers rely on low-level layers. The adjacent layers communicate with each other by defined protocols or interfaces.

The connector type between Data Repository Layer and Data Access Layer is Data Repository Connection. The Data Retrieval component connects to data stores and the Service Agent connects to services. The connector between Data Access Layer and Communicator Layer is in the type of Request-Response, the Communicator Layer sends requests to the Data Access Layer and gets responses with data, which follows the Client-Server architectural style. The communication between Communicator and Interactor is defined by the logic in Store component, by Method Call. And Connector from Presentation Layer to Interactor Layer is in the form of Method Call, e.g., an event listener function. From Interactor Layer to Presentation Layer the Observer pattern is applied as the connector, that the UI Manager in Presentation Layer subscribes to the states in Interactor Layer. When there are changes on the states the Store notifies the UI to update the view. The connectors between layers are described in Table 14.

Table 14: Connector types between Layers of the architectural pattern

Connector type	Relevant layers	Relevant internal components
Data Repository Connection	Between Data Repository and Data Access	Data Retrieval, Service Agent
Request-Response	Between Data Access and Communicator	Data Communicator, Data Retrieval, Service Agent
Method Call	From Interactor to Communicator, From Presentation to Interactor	Action, Data Communicator, UI Manager
Observer (Publish-Subscribe)	From Interactor to Presentation	State Store, UI Manager

5.1.2.2. Connectors within Data Access Layer

The components in Data Access Layer include Data Retrieval, Service Agent, and Data Processor. The connection type is that Data Retrieval and Service Agent call the methods of Data Processor, to make use of the functions to manipulate data. The connector in Data Access Layer is described in Table 15.

Table 15: Connector type in Data Access Layer of the architectural pattern

Connector type	**Relevant layers**
Method Call/Dependency	From Data Retrieval and Service Agent to Data Processor

5.1.2.3. Connectors within Communicator Layer

The components in Communicator Layer are Data Communicator and Data Helper. Data Communicator makes use of functions in Data Helper, so the Connector type is method call/dependency from Data Communicator to Data Helper. The description is presented in Table 16.

Table 16: Connector type in Communicator Layer of the architectural pattern

Connector type	**Relevant layers**
Method Call/Dependency	From Data Communicator to Data Helper

5.1.2.4. Connectors within Interactor Layer

The communication in this layer is that the Action triggered a dispatch to update the states in State Store. So the connector type can be defined as Dispatcher. Table 17 shows the description of the connector type.

Table 17: Connector type in Interactor Layer of the architectural pattern

Connector type	**Relevant layers**
Dispatcher	From Action to State Store

5.1.2.5. Connectors within Presentation Layer

There are dependencies between components in Presentation Layer, because the UI Manager relies on the Map Manager to create and manage map elements, the Map Manager relies on the Layer creator to create layers, and the layer creator relies on the Feature Styling to generate display elements for a layer.

Table 18: Connector type in Presentation Layer of the architectural pattern

Connector type	**Relevant layers**
Method Call/Dependency	From UI Manager to Map Manager, From Map Manager to Layer Creator, From Layer Creator to Feature Styling

5.1.3. Constraints

The architectural elements are well defined with the component and connector types. Based on this consequence, we set constraints on these components and their relations as a supplement of the pattern definition.

- The Presentation Layer does not handle the interaction process that needs to retrieve data from the server. It only concerns the display of map and other UI components, acceptance of user input and independent client-side interactions. The process of complex interactions has to be managed in Interactor and Communicator Layer.
- In principle each layer only communicates with the adjacent layer, to complete the display and interaction process. The exception is the display of layer without many interactions, e.g., the

display of a static basemap layer, in which case the Presentation Layer communicates with the Data Access Layer directly.

5.2. Representation of an Architectural Pattern

In this section a formal representation of the architectural pattern is presented, with the method proposed by Gamma et al. (Gamma et al., 1995). The Intent, Motivation, Applicability, Structure, Participants, Collaborations, Consequences and Related Patterns are described. The Implementation part of the pattern will be discussed in Chapter 6. In the Structure part, UML is used as the modeling language for graphic notation. The types of Package Diagram and Component Diagram are used to illustrate the layer structure and internal components structure.

5.2.1. Intent

Provide a template to easily build a geospatial web application.

5.2.2. Motivation

There are recurrent problems in designing a geo-web app. The complexity of spatial data affects the way to manipulate and visualize the data. And there are always requirements of rich interactions on the map. So an architectural pattern can be proposed to provide a reusable solution to address these problems.

5.2.3. Applicability

Use this architectural pattern when:

- The major purposes of the web application are to visualize a map and allow the user to interact with the map.
- There are requirements of complex user interactions with dependencies between each other.

5.2.4. Structure

Figure 20 presents the layer structure of the pattern. Figure 21 – 24 presents the component structure in Data Access Layer, Communicator Layer, Interactor Layer and Presentation Layer respectively. The package diagram and component diagram, as two standardized structural diagram types in UML are used for presentation. Package diagram can be used to illustrate groups of elements, and component diagram describes the organization of components in a system.

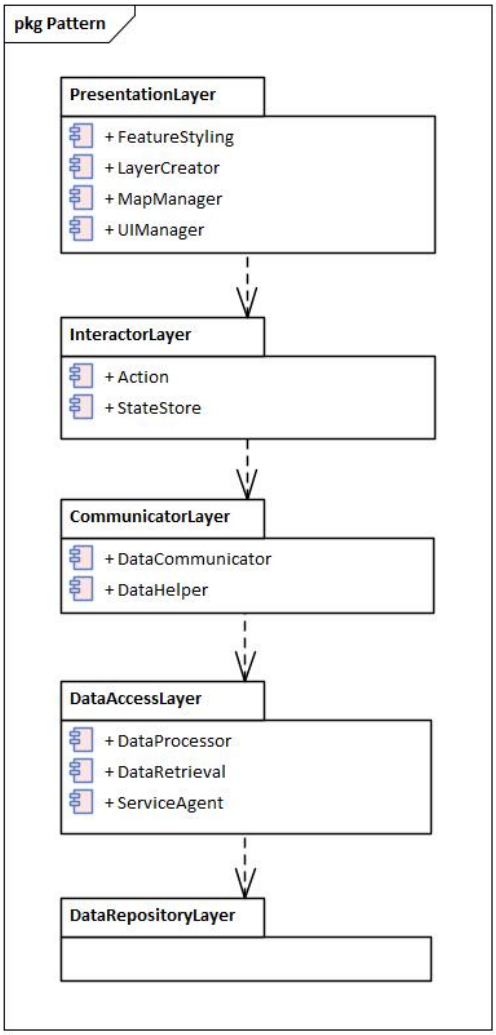


Figure 20: Layer structure of the architectural pattern by package diagram

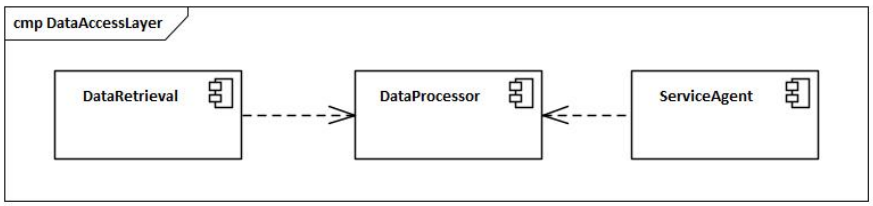


Figure 21: Components structure in Data Access Layer by component diagram

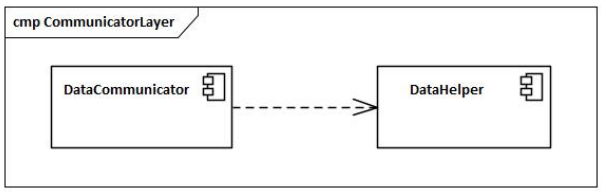


Figure 22: Components structure in the Communicator Layer by component diagram

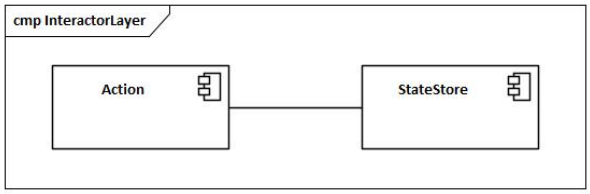


Figure 23: Components structure in the Interactor Layer by component diagram

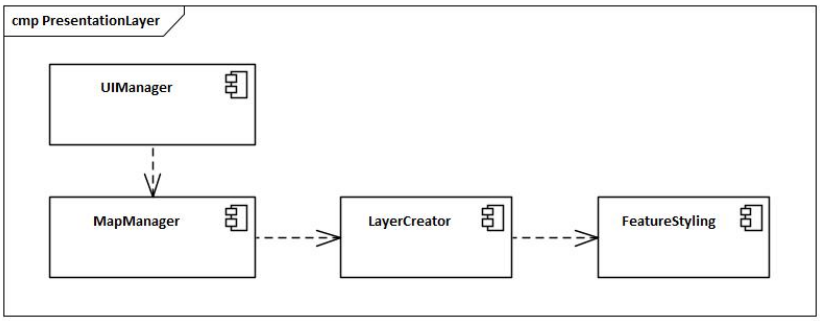


Figure 24: Components structure in the Presentation Layer by component diagram

5.2.5. Participants

- **Data Repository Layer** - Store and maintain the spatial and non-spatial data and service.
- **Data Access Layer** - Connect to Data Repository and access data.
  - **Data Retrieval** - Connect to the data repository and retrieve data.
  - **Service Agent** - Connect to service and retrieve data.
  - **Data Processor** - Manipulate the retrieved data.
- **Communicator Layer** - Send requests and get responses for data.
  - **Data Communicator** - Communicate with Data Access Layer to get data.
  - **Data Helper** - Manipulate the result.
- **Interactor Layer** - Manage the complex user interactions.
  - **Action** - Process the effects of user events and notify State Store.
  - **State Store** - Manage the state of the view.
- **Presentation Layer** - Manage the display of UI components, receive user input, and handle simple user interactions.
  - **UI Manager** - Manage the display elements.
  - **Map Manager** - Manage the display of the map element and operations on it.
  - **Layer Creator** - Manage the generation of the layer element and operations on it.
  - **Feature Styling** - Control the symbolizing of spatial features.

5.2.6. Collaborations

Collaborations in the pattern follow the scheme that high-level layers rely on low-level layers. The adjacent layers communicate with each other.

- In Presentation Layer, the UI Manager component depends on the Map Manager component to manage map elements. The Layer Creator contributes to the creation of layers of a map. And the Feature Styling is used by the Layer Creator to add styles to features. The user events on UI components trigger a method call to the Interactor Layer, to start an interaction workflow.
- In Interactor Layer, the Action receives messages from the Presentation Layer, and processes the interaction by communicating with Communicator Layer to ask for data. And the actions are dispatched to the State Store, to update the states. The UI Manager subscribes to the states. When there are changes in a state, it notifies the UI components to be updated.

- In Communicator Layer, when there is a method call from Interactor Layer, the Data Communicator sends requests to Data Access and get responses. The manipulation of the results is managed by a Data Helper component.
- In Data Access Layer, the Data Retrieval and Service Agent receives requests from Communicator Layer for data, and retrieves the data from data stores or services. The Data Processor component help process the data.
- In Data Repository Layer, the data stores and services provide interfaces to connect to the Data Access Layer.

5.2.7. Related Patterns

This architectural pattern makes use of some design patterns and architectural patterns as listed below:

- **Three-layer architectural pattern**

The layer structure of this pattern is based on the three-layer architectural pattern.

- **Client-server architectural pattern/style**

The client-server architecture is taken as a basis of the design.

- **Observer**

The relation between Presentation Layer and Interactor Layer makes use of the Observer pattern.

6. AN IMPLEMENTATION OF THE PATTERN

This chapter discusses an implementation of the architectural pattern. In section 6.1 the Pandviewer project is introduced, as the use case to be implemented. And then an implemented workflow is explained in 6.2, following the process introduced in 3.4. In 6.3 the result of the application is presented.

6.1. Pandviewer Introduction

Pandviewer (Buildingviewer) is a project as a part of the Self-Service-GIS (SSGIS) Platform of Kadaster(*Kadaster Labs*, n.d.), which aims at providing access to Kadaster's data and other open-source data to non-expert users(Rowland et al., 2020). The architecture of this platform is presented in Figure 25. Based on the databases, there are knowledge graphs to link data and information together, and APIs of the data services, such as SPARQL and Elastic Search are provided, by which SSGIS applications can be developed. There have been some developed applications and open source tools that can be integrated in SSGIS platform, e.g., the GeoData Wizard to allow users to upload a csv file and convert it to linked data(*LDWizard*, n.d.).

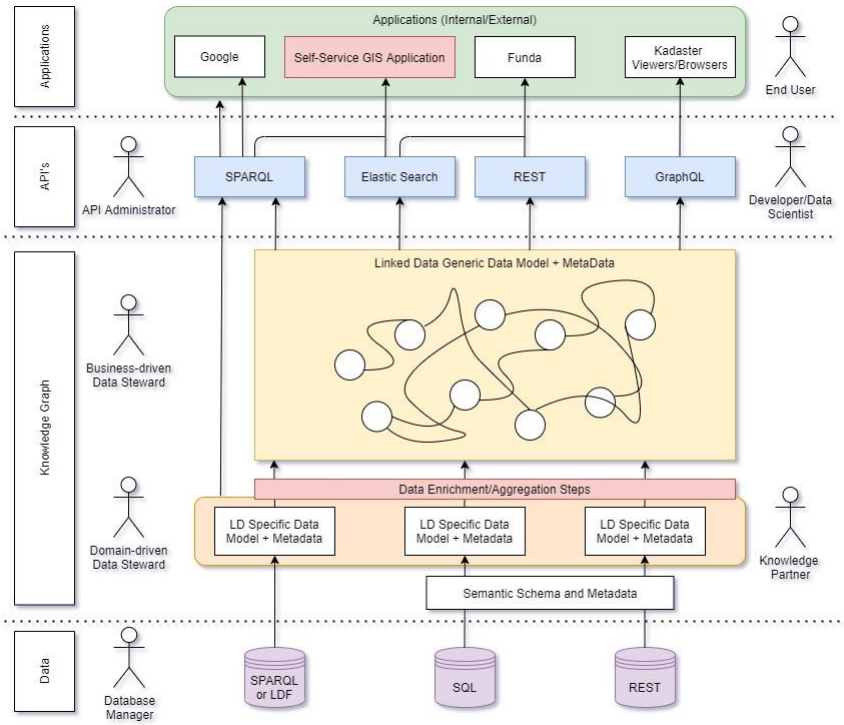


Figure 25: Self-Service GIS architecture of Kadaster(*Kadaster Labs*, n.d.)

The concept of Pandviewer was proposed as a data viewer, to enable the searching and viewing of building data. There are requirements for citizens to search for an interested building, and check the specific information. Kadaster provides datasets that include information on buildings in the Netherlands. A portal to allow the user to perform queries on these datasets on a map interface is of great concern.

The choice of Pandviewer as a typical example to implement the architectural pattern is because of the following reasons:

- The major requirement of this application is to visualize geospatial information.

- This application needs to use geodata with a complex structure.
- There need to be various interactions on the map and other components.

Pandviewer matches typical features of a geo-web app, which includes map display, map interactions, .etc. In this research, I follow the architectural pattern approach to implement Pandviewer as a version independent from the production version in Kadaster. The implementation process can validate the pattern and provide an implementation sample.

6.2. Implementation Workflow

6.2.1. Requirement Analysis

6.2.1.1. Purpose and user group

The intent of Pandviewer is to provide a map viewer of buildings in the Netherlands. The information of buildings, e.g., construction year, address and type is concerned. There are datasets from Kadaster with registrations of buildings, and Pandviewer provides a user-friendly view for these datasets.
The user groups of the application can be diverse. Residents can make use of the application to explore information about buildings. Besides, it can serve as a demonstration tool for professionals.

6.2.1.2. Functionality

The main functionality of Pandviewer is to query for a building by location or address, and then display the feature on the map with information on the pop-up. The concrete functionalities are discussed below under the category of Map Display and Component Interaction.
The user interface of Pandviewer contains a map, to display the background map of the Netherlands and query results of the building. There are requirements that the user may prefer to view the result above a standard background map with roads, rivers, etc. or an aerial image background map that is closer to the realistic features. So two basemap layers can be offered as options. Besides basemap layers, a feature layer is required to display the query result. For each query there is only one result, so this layer only contains one feature. The map layer structure of Pandviewer is presented in Table 18.

Table 19: Map layer structure of Pandviewer

Layer	**Description**
Standard background layer (basemap layer)	Static basemap layer to display background map of the Netherlands
Aerial image layer (basemap layer)	Static basemap layer to display aerial image map of the Netherlands
Feature layer	Interactive thematic layer to display query result of building

There are interactions on the map and other components. The user can query a building in two ways. The first way is to right-click on the map to find the building at that location, and the second way is to input the postcode and house number to find the building. After the query the result is displayed on the map as a feature layer, and the user can view the information of that building on a pop-up by moving the mouse over it. The pop-up contains information of address, construction year, registration name, status and type of a building, hyperlinks to navigate the user to the data source page. The interactions on Pandviewer are described from Table 20 to Table 23, according to the classification discussed in 4.1.

Table 20: Map navigation interactions of Pandviewer

Interaction	**Description**
Scroll to zoom	Zoom in or zoom out of the map by the scroll wheel

Drag to pan	Pan to any area by dragging the map
Zoom control to change the zoom	Zoom in or zoom out by the zoom control on the map
Zoom and pan to fit query result	When there is a query result, the map will be panned to that area, and the scale will be changed to fit the visualization of the object.

Table 21: Search and filter interactions of Pandviewer

Interaction	**Description**
Coordinate search	Search for a building by right-clicking on a location on the map.
Address search	Search for a building by type in the postcode and house number and click the 'search' button.

Table 22: Information retrieval interactions of Pandviewer

Interaction	**Description**
Pop-up display	Display the attributes of the building object by moving the mouse on the feature to show the pop-up. Hide the pop-up by clicking on the map.

Table 23: Map configuration interactions of Pandviewer

Interaction	**Description**
Reset	To clear the feature layer on the map by clicking on the header.
Layer switch	To switch the basemap layer by the layer switch control on the map.

6.2.1.3. User interface design

Based on the requirement and functionality analysis, a conclusion can be made on the required UI components of the application. There should be a header including the title of the application. A map is needed to display background map and result feature. And to perform the coordinate search functionality, two input boxes for postcode and house number, and a button for starting search are required. The design of the user interface layout is illustrated in Figure 26. The map takes up the most space, to ensure a user-friendly visualization. On the left of the screen, there is a header, introduction text, input boxes and a search button.

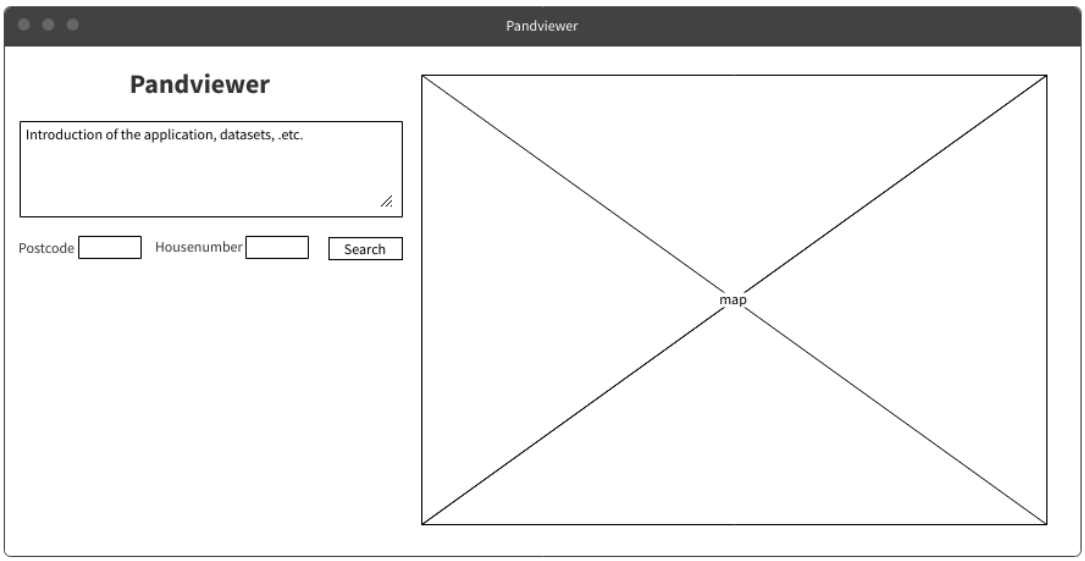


Figure 26: User interface design of Pandviewer

6.2.2. Data Collection

6.2.2.1. Basemap layer data source

The required data for the application are the source of the basemap layers and that of building objects. The basemap layers are relatively static and the suitable data type for basemap layers is the tiled image. According to the requirements, the following image sources are selected.

- **Luchtfoto / PDOK (Open)**(*Introductie - PDOK*, n.d.-a)
  It a national aerial photograph produced annually, produced by the authorities participating in samenwerkingsverband Beeldmateriaal(*Home | Beeldmateriaal Nederland*, n.d.), with a resolution of 25 centimeters.
- **Basisregistratie Topografie Achtergrondkaarten (BRT-A)**(*Introductie - PDOK*, n.d.-b)
  It is a background map of the Netherlands as a product of BRT. The map is provided by Kadaster and made available by PDOK.

These data sources are open data with provided geo services. To make use of these image data no deployment of data is needed. The service types and URLs are presented in Table 24.

Table 24: Image services for basemap layers of Pandviewer

Image service	Service type	URL
BRT Achtergrondkaart Standaard V2	WMTS	https://service.pdok.nl/brt/achtergrondkaart/wmts/v2_0?request=getcapabilities&service=wmts
Luchtfoto Beeldmateriaal / PDOK 25 cm RGB	WMTS	https://service.pdok.nl/hwh/luchtfotorgb/wmts/v1_0?request=GetCapabilities&service=wmts

6.2.2.2. Feature layer data source

The feature layer is to display the query result and should enable pop-up interactions. Vector data is the appropriate type for the data source. To cover the required information including building geometry, address, construction year, status, type and name, the following data source can be used.

- **Basic Registration Addresses and Buildings (BAG)**(*BAG - Kadaster.Nl Zakelijk*, n.d.)
  This dataset contains registrations of addresses and buildings of the Netherlands(*Cat. BAG 2018*, n.d.). It can provide the major information for a building, such as geometry, address including street name, house number, postcode and city name, construction year and status. The data model of this dataset is presented in Figure 27, which illustrates the properties of each object and how it relates to each other.
- **Basic registration Large-scale Topography (BGT)**(*Basisregistratie Grootschalige Topografie (BGT)*, n.d.)
  BGT is a dataset concerning topographical objects of the Netherlands including buildings, roads, railways, waterways, .etc(*Basisregistratie Grootschalige Topografie Gegegevenscatalogus BGT 1.2*, n.d.). The required information for Pandviewer is status of a building.
- **Basic Registration Topography (BRT)**(*BRT - Kadaster.Nl Zakelijk*, n.d.)
  This dataset concerns topographical objects at different scale levels. The registration of buildings is also included. To complement the information from BAG and BGT, BRT can provide the type and name/label information of a building.

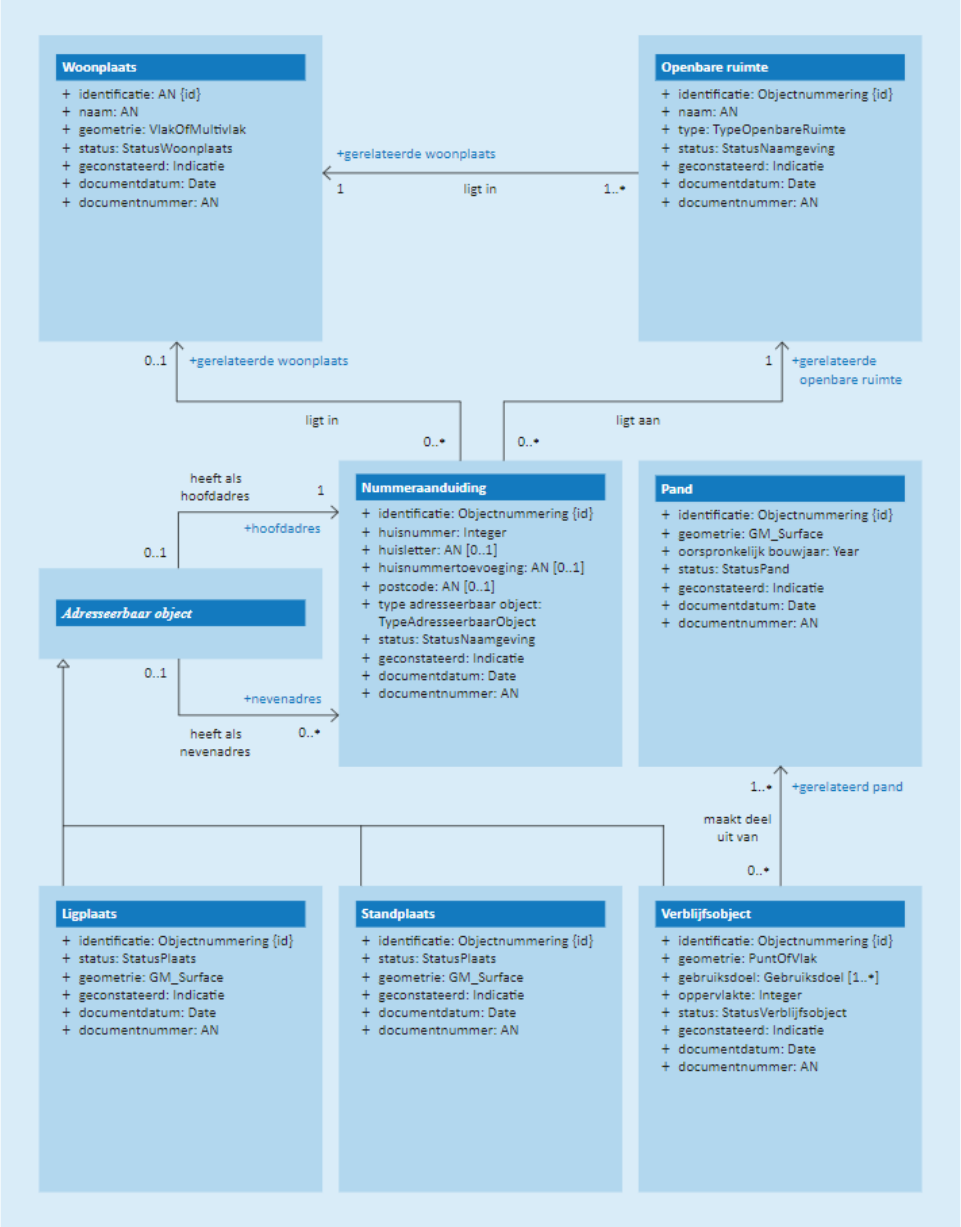


Figure 27: Conceptual model of the BAG(*Conceptueel Model*, n.d.)

These datasets are available as linked data and can be accessed with endpoints. There are RESTful APIs for accessing the query services, including SPARQL and Elastic Search. As a result, the deployment of databases for this application is not needed. The services of BGT and Kadaster Knowledge Graph can be used. The BGT provides SPARQL service to query for BGT objects, and Kadaster Knowledge Graph provides SPARQL service to link BGT objects with BAG and BRT objects. The service types and endpoints are listed in Table 25.

With these services, the concerned questions can be translated to SPARQL queries and implemented. SPARQL is a standard query language for linked data(*What Is SPARQL? | Ontotext Fundamentals Series*, n.d.). Extensions of GeoSPARQL enables spatial query, to answer questions related to locations.

Table 25: Data services for feature layer of Pandviewer

Data service	**Service type**	**Endpoint**
BGT	SPARQL	https://api.labs.kadaster.nl/datasets/kadaster/bgt/services/bgt/sparql

Kadaster Knowledge Graph	SPARQL	https://api.labs.kadaster.nl/datasets/kadaster/kg/services/default/sparql

6.2.3. Architecture Design

In this sub-section the process of a conceptual design of the application with the architectural pattern is explained. The architecture is derived, based on the defined components and connectors.
The user requirements of Pandviewer are discussed in 6.2.1, with which the necessities of pattern components can be analyzed. The architectural pattern suggests a 5-layer architecture, in which all the layers make sense in this case. The layer structure for Pandviewer is shown in Figure 28, indicating that Databases and Data Service Layer are Data Repository Layer and Data Access Layer in the pattern. And the usefulness of components within layers is discussed in Table 26.

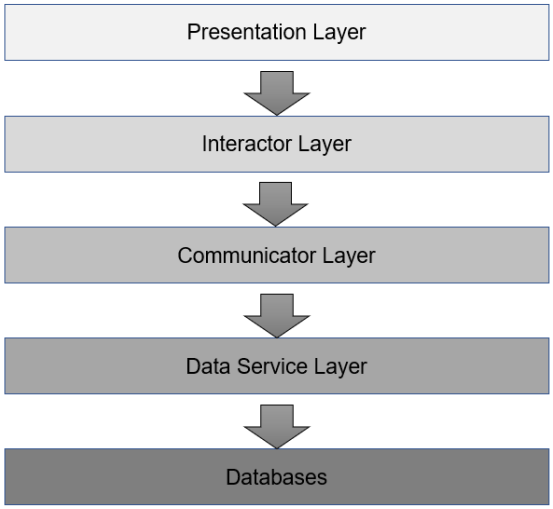


Figure 28: Structure of layer components of Pandviewer architecture

Table 26: Analysis of necessity on components within layers for Pandviewer architecture

Component	**Necessary or not**	**Explanation**
Data Retrieval	No	A Service Agent component can take the responsibility to retrieve data from services.
Service Agent	Yes	The connection to data services and execution of queries should be managed by this component.
Data Processor	No	There is no need to perform addition calculation on data.
Data Communicator	Yes	This component is necessary to send query requests, and return the results.
Data Helper	Yes	The results should be manipulated to convert the format of geometry, for feature display. Also there should be operations to reorganize the results.
Action	Yes	The circle of a searching interaction needs to be managed by the Action component to control the behaviors.
State Store	Yes	The state of the application, which influences the presentation, should be controlled by the State Store.

UI Manager	Yes	It is a necessary component to control the render and display of UI components and handle user inputs.
Map Manager	Yes	There is a requirement to display a map, and include map interactions.
Layer Creator	Yes	There is a need to create different types of layers on the map.
Feature Styling	No	The feature styling task can be included in the Layer Creator component, since the symbolization is in a simple way with a consistent style.

After the determination of the required components, a further step is to define the concrete responsibility for each component, according to the required functionalities. In Table 27 the responsibility is described.

Table 27: Component responsibilities in the architecture of Pandviewer

Layer	**Component**	**Responsibility**
Data Service Layer	Query Agent (Service Agent)	o Connect to databases. o Execute the query for coordinate search and address search. The query includes tasks to find the target building given the parameters of coordinates or a pair of postcode and house number, find the attributes of that building, and filter out invalid records.
Communicator Layer	Data Communicator	o Request for the building object data with parameters and return the result. o Call the Data Helper functions for manipulation.
	Data Helper	o Manipulate the retrieved data for geometry format conversion and feature reorganization.
Interactor Layer	Action	o Hold the behaviors of the application. According to two types of search, there are actions of Search Start, Search Success and Search Error. And the Reset action and Zoom Change action are required. o Perform operations of these actions. For the Search Start action, it interacts with Communicator Layer to retrieve data. o Dispatch the actions to change the states.
	State Store	o Hold the states of the application, including the query parameters, fetching states, query result and zoom level. o When there are changes it notifies the presentation layer, to update the map display, e.g., change the zoom, draw the feature and display it, clear the feature.
Presentation Layer	UI Manager	o Define the structure and style of UI components. o Define the way to get user input and bind the user event to actions. For example, The Address Search Start action is triggered by clicking a button, and the text in input boxes is passed as the query parameters.

	Map Manager	○ Manage the map display functions, including the initialization of map elements, simple interactions on the map including scroll to zoom, drag to pan, zoom control, layer switch and pop-up, and other behavioral functions. ○ Bind the right-click event on the map to the corresponding actions. ○ Subscribe to the state to update the view, to implement result display, zoom change, .etc.
	Layer Creator	○ Generate the layers on the map. The generation of basemap layers retrieves image data from the map service directly, and the generation of the feature layer uses the feature data from the states, defines the style of the polygon feature, and adds a pop-up with information to the feature.

According to the analysis, the architecture of Pandviewer is presented in Figure 29. Besides the components discussed above, there is a Map Service component as an external component providing image data for basemap layers. The relation between layers and components follows the connector types discussed in 5.1.2. The exception is that there is direct communication between Presentation Layer and Data Service Layer, to handle the basemap display from map services.

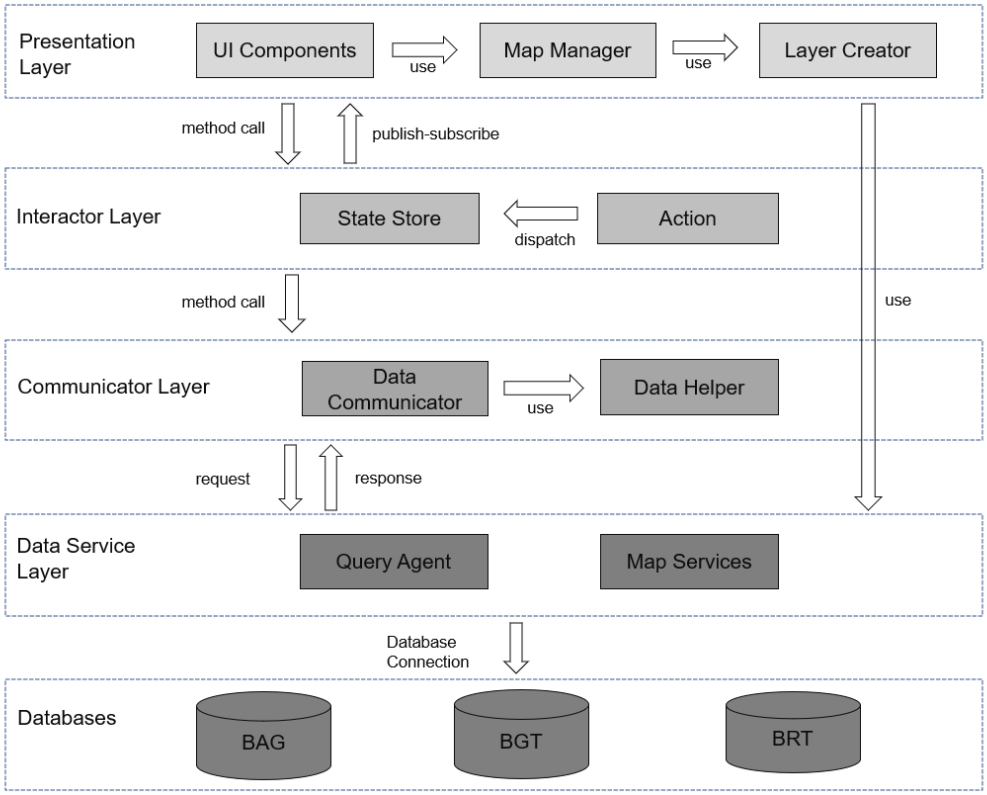


Figure 29: Architecture design of Pandviewer

6.2.4. Techniques Survey

The selection of techniques is a concern for the implementation. The following techniques are selected for the developing of Pandviewer.

6.2.4.1. Data storage mechanism

According to the data source of Kadaster linked datasets, Kadaster Triple Store is available as a tool to store and organize datasets(*Kadaster Data Science*, n.d.). In addition to storing linked data and provide services, it can also be used to hold queries and provide RESTful APIs for these queries. For Pandviewer we use Triple Store for the majority of back-end works, including Databases and Data Service Layer, since it can fulfill the storage and query creating, testing and execution of linked data, and the required datasets are available on this site.
I use SPARQL as the query language on these datasets, as it is a standard semantic query language and supported by Triple Store.
The service type for basemap layers is Web Map Tile Services (WMTS), which is an OGC standard for tiled map service and suitable for basemap display. The selection is based on the available service type of these data sources, which is discussed in 6.2.2.

6.2.4.2. Programming language

To build the application, the coding is only in the front-end since the back-end is implemented by Triple Store and external services.
I use TypeScript as the programming language, which is a superset of JavaScript, adding the feature of optional types(GitHub, 2020). JavaScript/TypeScript is the only standard programming language for web front-end developing. And compared with JavaScript, TypeScript has advantages like type check, and better supports for OO programming, which makes it an appropriate choice for our implementation.

6.2.4.3. Framework

The back-end of Pandviewer is supported by Triple Store, which has its own mechanism to process the data store and query execution tasks.
And the front-end uses the React framework, which provides a solution to manage UI components for an application(*React – A JavaScript Library for Building User Interfaces*, n.d.). React fits the requirements of our design because it not only helps create UIs for user-friendly view, but also allows interactions on these components. For functional components in React, UI components can be created by defining a set of functions. With the tool of Hooks, the interaction process can be implemented. The selection of React is based on the following reasons:

- Developing habits of the team.
- Advantages in creating and managing interactive UI components.
- Various Hooks to provide functions on UI.
- Great support of TypeScript.
- Convenience in developing single-page applications.

6.2.4.4. Library

A web mapping library is required for Pandviewer, to easily create interactive web maps. There are several available open-source or commercial web mapping libraries, such as OpenLayers, Leaflet, and Mapbox GL. Leaflet is selected for this application, as it is a lightweight library offering high performance. The web mapping requirements of Pandviewer, as is discussed in 6.2.1, can be easily handled by Leaflet.

6.2.5. Application Development

The developing of Pandviewer includes creating queries in Triple Store and coding for the front-end. The implementation of the architectural elements is discussed in this sub-section.

6.2.5.1. Back-end work

The Triple Store provides the functions of Query Agent component. Queries can be formed and tested in Triple Store. These queries target to find the building which meets the condition, with many attributes, to answer the following questions:

- What is the building whose geometry covers the point with the specified coordinates and what are the address, construction year, status, type, label status of that building?
- What is the building whose postcode and house number are consistent with the input value and what are the address, construction year, status, type, label status of that building?

Based on data models of datasets, the conceptual model of objects relevant with Pandviewer query in Knowledge Graph is presented in Figure 30.

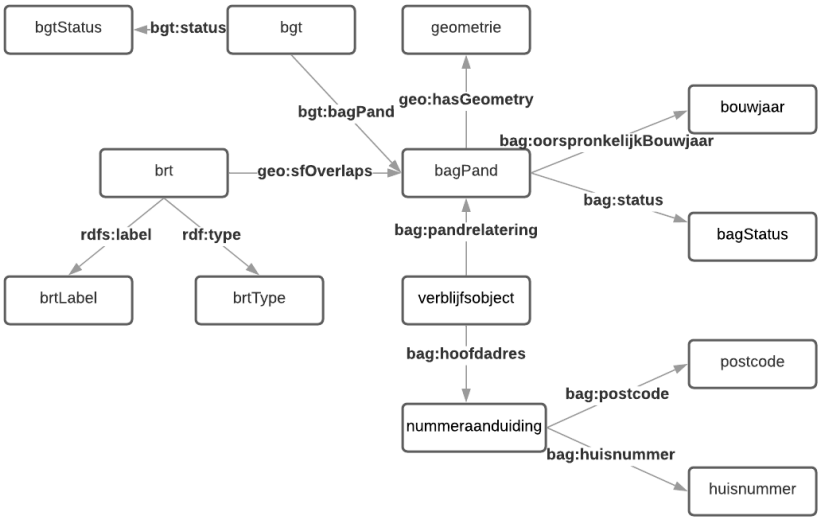


Figure 30: Conceptual model of objects relevant with Pandviewer query

The queries with the two questions are created in SPARQL language. And RESTful APIs of the queries are provided, as described in Table 28.

Table 28: Query APIs of Pandviewer

Query purpose	Query link	API
Coordinate Search	https://data.labs.kadaster.nl/jiarong-li/-/queries/PandviewerTest/22	https://api.labs.kadaster.nl/queries/jiarong-li/PandviewerTest/run?
Address Search	https://data.labs.kadaster.nl/jiarong-li/-/queries/PandviewerSearch/8	https://api.labs.kadaster.nl/queries/jiarong-li/PandviewerSearch/run?

6.2.5.2. Front-end development

The Communicator Layer, Interactor Layer and Presentation Layer constitute the front-end of the application. The separation of layers and components is declared by folders and files. Figure 31 illustrates the file structure of Pandviewer codes.

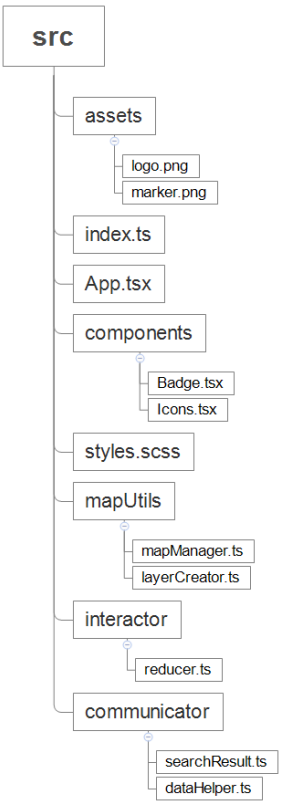


Figure 31: Folder structure of Pandviewer codes

The Communicator Layer is managed by the communicator folder. The Data Communicator is implemented by functions in searchResult.ts. The async fetch functions make it possible to send AJAX requests. And it uses the GET method of HTTP requests to access data from the query APIs. Functions in dataHelper.ts implements the Data Helper component, to manipulate the result.

The Interactor Layer is managed by the interactor folder, with a file reducer.ts and some functions in App.tsx. The state management of Pandviewer is implemented by React Hooks, as a set of tools to manipulate states. The actions and states are declared in reducer.ts, in which a reducer function is defined, to set the rule to update the states with actions. The operations to interacting with Communicator Layer actions are defined by the useEffect Hook.

The UI components are managed by the App.tsx, components folder, and style.scss, which defines the structure of components, provides component templates and defines styles respectively. App.tsx also implements getting user input and binding user events to actions. The useEffect Hook serves like the Observer pattern, that the application can monitor the state of Search Results, and to update the display when there are changes on results. The mapUtils folder implements the Map Manager and Layer Creator components, to provide functions to display the map. Simple interactions like pop-up display are implemented by it. Also the right-click event on the map is defined and connect to the functions in Communicator.

6.3. Result

The result of Pandviewer is presented by screenshots to illustrate the consequences of the main functionalities. Figure 32 shows the initial interface of Pandviewer. Figure 33 shows the result of an

address search. Figure 34 shows the result of a coordinate search. Figure 35 shows the result to display a pop-up information of an object.

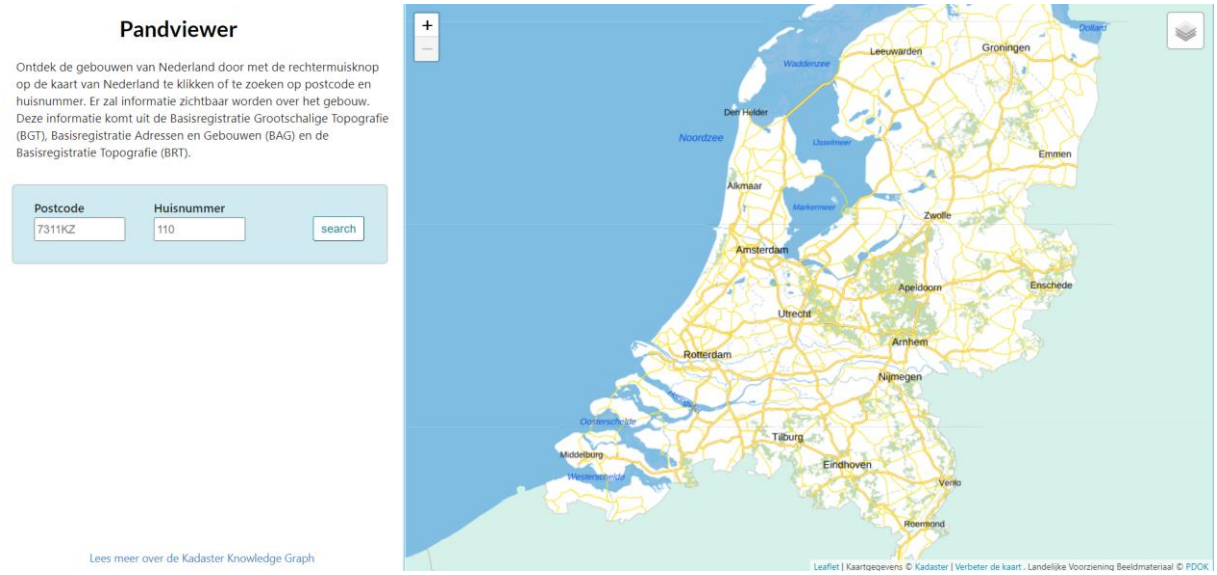


Figure 32: Initial User Interface of Pandviewer

Figure 33: Search for a building by postcode and house number in Pandviewer

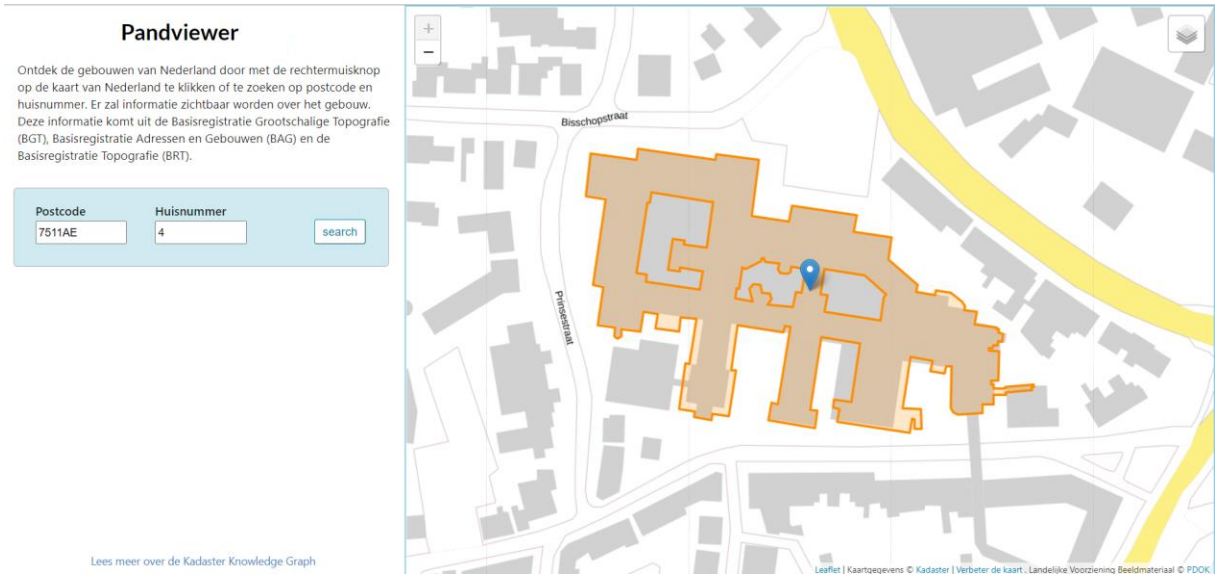


Figure 34: Search for a building by the location in Pandviewer

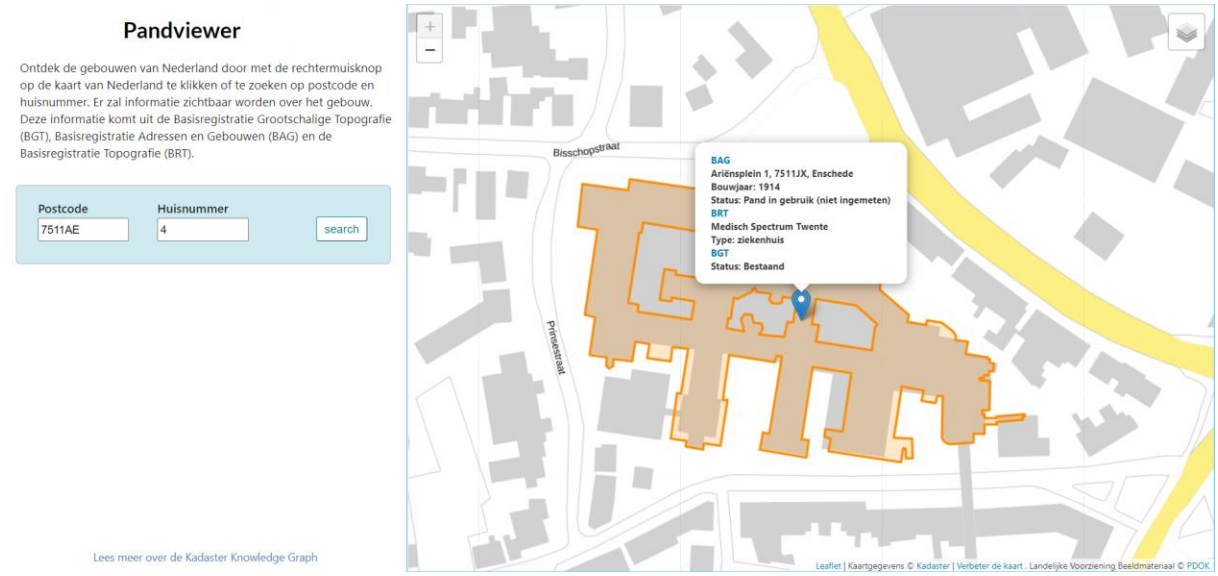


Figure 35: Display the pop-up information of a building in Pandviewer

6.4. Reflection

In this section a reflection on the implementation of the architectural pattern is given, together with an evaluation according to the OO principles.

6.4.1. Discussion on implementation

The approach and an example of implementation is explained in chapter 3 and the previous sections of this chapter. The pattern provides a template for the architecture of a geo-web app. To make use of the pattern, the implementation process is followed.

The requirement analysis process can take the classification method discussed in 4.1, to consider the map display and component interaction functionalities. Pandviewer is with a simple user requirement, which can be described by this approach clearly. An application with complex functionalities can use a use case diagram to discuss the role of users and use cases and their collaborations.

The data collection step needs to determine the data sources and the data store mechanism. For Pandviewer the data source is limited to Kadaster's datasets or relevant open-source datasets. And then the combination of BAG, BGT and BRT is determined based on the required information. Also the map services are selected from the PDOK platform. Pandviewer uses external databases, which excludes steps of data cleaning and data deployment. And it makes use of the query services from Kadaster Knowledge Graph. For the data collection process of an application using internal databases, there are design choices on the data storage mechanism, which should be based on the requirements on retrieving data. A regular workflow for data collection is presented in Figure 36, with comments of concerns for some steps.

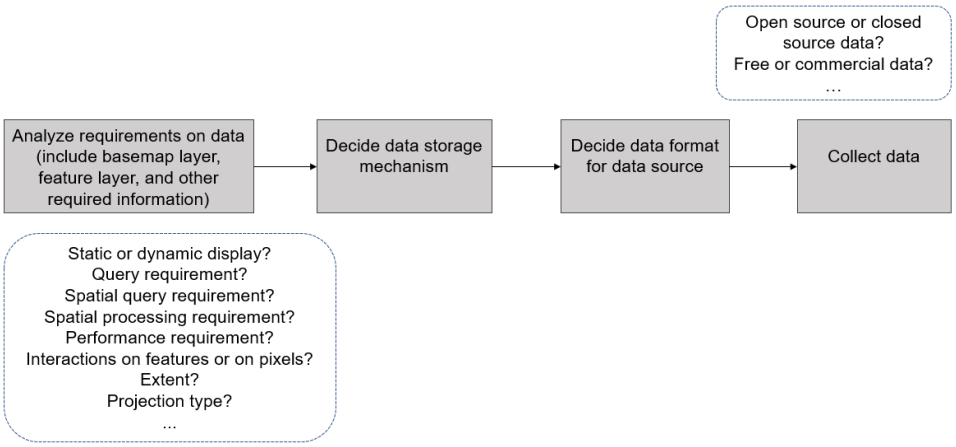


Figure 36: Workflow of the data collection step of the implementation process

The architecture design of the application applies the pattern approach. The pattern is in a Layered view, so the layer structure is decided first. Pandviewer follows the same layer structure of the pattern. According to the specific requirement of an application, the layers can also be modified, e.g., adding a business layer if there are complex business logic or adding a service layer if the system provides services for external systems. And then the internal components of the pattern are discussed about their necessity. Some components are excluded for Pandviewer. When designing an architecture, components with too many responsibilities can be divided, those with few tasks can be merged to the other components, and those not necessary in the design can be excluded, to ensure a balanced architecture. After analyzing the required components, an architecture can be built, with the redefined components and connectors.

The techniques survey is to find the appropriate techniques to help implement the application. For Pandviewer, Triple Store, SPARQL, TypeScript, React and Leaflet is selected as a set of tools. The architectural pattern is an abstract scheme and does not provide concrete functions for implementation. And there are few limitations on techniques and tools when applying the pattern. Usually a state management tool is required for the Interactor Layer, and it is recommended to use a front-end framework, with the supported state management tools. In Table 29 we present a description of the most popular front-end frameworks: Angular, React and Vue, with the corresponding state management plans. Even though the structure of the Interactor Layer is mainly derived from Redux and Flux, other state management tools show a similar pattern to control states by the store. When implementing, the design of Interactor can be based on this pattern structure with modifications according to the specific state management plan. The selection on framework can be based on complexity of the system and preference of the developer. A web mapping library is always required, to provide functions for Map Manager, Layer Creator and Feature Styling. Basic functions are supported by commonly used libraries, such as OpenLayers, Leaflet and Mapbox GL. For advanced functionality, there needs to be a survey on these libraries and their plug-ins to fit the requirements.

Table 29: Description of Angular, React and Vue Framework

Framework name	Description	State management tools
Angular	Angular is a Model-View-Whatever (MVW) framework, which divides a system into three components.	NGRX, NGXS
React	React is a library to create UI elements. Sometimes a third-party library is required to implement functionalities.	Redux, Context
Vue	Vue uses the MVC architecture. It is a lightweight framework compared with Angular and React.	VueX

After finalizing the architecture design and techniques survey, the application can be developed. The logic separation of components can be mapped to the physical separation. The development of Pandviewer is divided into back-end works and front-end programming. And in the front-end the separation of responsibilities is reflected in the structure of folders and files. Pandviewer uses external datasets and services, so there is no need to deploy databases and services. For applications using internal data stores, there should be additional steps to deploy data and creating services according to the choice and design from previous steps.

6.4.2. Consequences of the pattern

The consequence of the pattern is the realization of low coupling and high cohesion. The applying of layer structure enables horizontal and vertical divisions of responsibilities. There is low coupling between layers. Each layer communicates with the adjacent layer by defined interfaces or protocols, which enables the reuse of components. For example, the Data Service Layer of Pandviewer is defined independently, as queries maintained by Triple Store. The APIs of queries are exposed, to be used by the communicator component. The flexibility is reflected in that these APIs can be used by other systems, and queries are easy to change, without changing the application code. There is high cohesion among the internal components within a layer. As an example, the Action and State Store components of Pandviewer highly depend on each other, to exchange information and implement the interaction process.

7. CONCLUSION AND RECOMMENDATIONS

In this chapter a summary of the research is given. In section 7.1 there is a conclusion on the research outcome, with answers to the research questions. Section 7.2 and 7.3 analyze the contributions and limitations of the research respectively. And in section 7.4 I provide recommendations for future work.

7.1. Research Outcome

The major outcome of the research is the proposal of an architectural pattern for developing a geo-web app. The common concerns of this kind of application analyzed and concluded. And then an architectural pattern is designed to addressing these concerns. The validation of the pattern is managed by an implementation. Finally the consequence of the pattern is derived. The objective of the research is finished. Below I give the answer to each research question.

7.1.1. Common concerns identification

i. What are the desired requirements and functionalities of a geo-web app?

The requirements of a geo-web app include analysis and visualization of geodata. The common functionalities are map display and component interactions. Map display involves the management of map layers and other map elements. And component interactions can be classified into types of map navigation, search and filter, information retrieval and map configuration.

ii. What are the common implementation elements of a geo-web app?

The common implementation elements of a geo-web app can be derived from the required functionalities. Based on the result of functionality, MapManager, LayerCreator, FeatureStyling, Interactor, DataRetrieval and DataProcessor are concluded as commonalities of the problems. For each commonality, there can be variations of implementation, e.g., generating different types of layers by LayerCreator. The commonalities and variations describe the problem and constitute the implementation elements of geo-web apps.

7.1.2. Architectural pattern design

i. How to construct the implementing elements for establishing an architectural pattern?

An architectural pattern is made up of pattern elements. The perspective to consider these elements concerns. The different perspectives are defined as architectural viewtypes. For designing the pattern, I pick up the Component-and-Connector view and the Layered view, as the approach to discuss and describe the pattern elements.

ii. What are the required architecture elements, relations between them and constraints on them?

The architecture elements are described as component types, connector types and constraints. The comprehensive definition of these elements is discussed in 5.1. There are divisions of 5 layers based on the responsibility levels. And within each layer the required components are defined. Then the connector types between layers and between components are determined, indicating how the elements communicate with each other. Finally there are constraints to set rules to the elements and their relations.

iii. What is the appropriate approach to representing the architectural pattern?

The representation of the architectural pattern uses the format proposed by Gamma et al. (Gamma et al., 1995). There are intent, motivation, applicability to give an overview of the pattern, and structure, participants, collaborations to present the pattern content. The consequences and implementations are required to indicate the result of applying this pattern and the approach to implement it. And related

patterns should be mentioned. In 5.2 I give a representation of the architectural pattern, and the implementation and consequence part is discussed in 6.4.

7.1.3. To implement the pattern by developing an application based on this template.

i. What is the workflow to develop an application applying the architectural pattern?

The implementation process is defined based on the WebGIS development cycle, with steps of requirement analysis, data collection, architecture design, techniques survey and application development. The architectural pattern is applied in the architecture design process, during which the architectural elements of the system are defined. For other steps, the recommended implementation approach is also given with the sample of Pandviewer and the reflection.

ii. What are the required types of techniques in the implementation, and what are the principles to select the techniques?

The techniques for implementation always include data storage mechanism, programming language, framework, and library. The selection of techniques is based on the user requirements, the platform, and the developing habits of a developer or a team. There can be dependencies between choices of techniques. The architectural pattern does not provide concrete functions nor set limitations to the techniques.

iii. What is the consequence of applying the architectural pattern?

The applying of the architecture pattern can achieve loose coupling and high cohesion of the system. The responsibilities can be clearly divided, which improves the reusability and maintainability of the application.

7.2. Contributions

This research starts from the analysis of common problems of geo-web apps. The conclusion of the required functionality can be useful for research on geo-web apps, as a perspective to classify the requirements.

The major consequence of the research is the proposed architectural pattern, which can serve as a clean template for developing geo-web apps. This pattern is reusable and independent from the developing techniques. Besides, a sample of implementation is given, with a clear guidance about applying this architectural pattern.

This research proves the possibility of adopting an architectural pattern approach to the GIS domain. There are explorations about combining the pattern to GIS developing, but most of them focus on design patterns, e.g., the recommendations on applying Composite pattern to represent geo objects(Gordillo et al., 1999). The architectural pattern differs from the design pattern in that it provides a complete template for building a system. To introduce architectural patterns especially for geo-web apps, developers unfamiliar with GIS domain can use this approach to build an application in a convenient way. A high level of reusability can be achieved with this method.

A workflow to derive an architectural pattern is also introduced in this research. The approach to design an architectural pattern is seldom mentioned in previous studies. A pattern is usually proposed by experts from their experiences and inspirations. This research introduces a method to combine the commonality and variability analysis with the architectural viewtypes, to conclude the common concerns and derive a pattern step by step. It provides a reference to build an architecture pattern for a system in a specific domain.

7.3. Limitations

There are limitations on the process of common concern identification, which focuss on a set of geo-web apps, especially Toponamenzoeker. These geo-web apps are partial to the interactive web mapping, which makes the derived common concerns incomplete. Some concerns, such as editing a geo object with a web app, are not covered. As a result, the pattern approach cannot fit all kinds of geo-web apps.

In the process of determining pattern elements, there can be more surveys on existing systems and relevant literature, to make convincing selections on the component and connector types.

The architecture pattern follows a traditional client-server style. The innovative architecture types, such as cloud computing discussed in 2.3.3 are not reflected.

There are not sufficient evaluations on the architectural pattern, because there is a lack of reference in the method of evaluation on patterns. This study uses an implementation to validate the pattern and discuss the consequences of applying the pattern by some OO principles.

7.4. Recommendations for Future Work

The further work of this research can be the improvement of the proposed architectural pattern. Based on the common concerns and architectural viewtypes, there can be more research on the existing systems, to enumerate the possible types of components and connectors. And then by comparison between these types from the specified requirements, the element types can be specified. A better solution as the improved pattern can be achieved by this way.

And there can be future work on proposing architectural patterns for a wider scope of the geo-web app, or a different type of geo-web app. The concerns can be derived with these different focuses, and the workflow to design a pattern introduced in this research can be taken.

LIST OF REFERENCES

About Us - Johns Hopkins Coronavirus Resource Center. (n.d.). Retrieved July 6, 2021, from https://coronavirus.jhu.edu/about

Agrawal, S., & Gupta, R. D. (2017). Web GIS and its architecture: a review. *Arabian Journal of Geosciences*, *10*(23), 1–13. https://doi.org/10.1007/s12517-017-3296-2

Alesheikh, A. A., & Helai, H. (2002). Web GIS : Technologies and Its Applications. *Symposium on Geospatial Theory, Processing and Applications*, *15*(August), 1–9. www.w3.org

Application Architecture Guide - Chapter 12 - Data Access Layer Guidelines - Guidance Share. (n.d.). Retrieved June 29, 2021, from http://www.guidanceshare.com/wiki/Application_Architecture_Guide_-_Chapter_12_-_Data_Access_Layer_Guidelines

Application Architecture Guide - Chapter 9 - Layers and Tiers - Guidance Share. (n.d.). Retrieved July 17, 2021, from http://www.guidanceshare.com/wiki/Application_Architecture_Guide_-_Chapter_9_-_Layers_and_Tiers

artima - Erich Gamma on Flexibility and Reuse. (n.d.). Retrieved July 31, 2021, from https://www.artima.com/articles/erich-gamma-on-flexibility-and-reuse

Avgeriou, P., & Zdun, U. (2005). *Architectural Patterns Revisited-A Pattern Language*.

BAG - Kadaster.nl zakelijk. (n.d.). Retrieved August 8, 2021, from https://www.kadaster.nl/zakelijk/registraties/basisregistraties/bag

Basisregistratie Grootschalige Topografie (BGT). (n.d.). Retrieved August 8, 2021, from https://www.digitaleoverheid.nl/overzicht-van-alle-onderwerpen/basisregistraties-en-stelselafspraken/inhoud-basisregistraties/bgt/

Basisregistratie Grootschalige Topografie Gegegevenscatalogus BGT 1.2. (n.d.). Retrieved August 8, 2021, from https://docs.geostandaarden.nl/imgeo/catalogus/bgt/

Béjar, R., Latre, M. Á., Nogueras-Iso, J., Muro-Medrano, P. R., & Zarazaga-Soria, F. J. (2009). An architectural style for spatial data infrastructures. *International Journal of Geographical Information Science*, *23*(3), 271–294. https://doi.org/10.1080/13658810801905282

BRT - Kadaster.nl zakelijk. (n.d.). Retrieved August 8, 2021, from https://www.kadaster.nl/zakelijk/registraties/basisregistraties/brt

Burrough, P. A. (1986). Principles of geographical information systems for land resources assessment. In *Principles of geographical information systems for land resources assessment.* Clarendon Press, Oxford; Monographs on Soil & Resources Survey, 12. https://doi.org/10.1097/00010694-198710000-00012

Catalogus BAG 2018. (n.d.). Catalogus BAG 2018. Retrieved August 8, 2021, from https://imbag.github.io/catalogus/

Chapter 9 Databases | Introduction to Web Mapping. (n.d.). Retrieved May 5, 2021, from http://132.72.155.230:3838/js/databases.html

Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003). Documenting software architectures: Views and beyond. *Proceedings - International Conference on Software Engineering*, *June*, 740–741. https://doi.org/10.1109/icse.2003.1201264

Coetzee, S., & Rautenbach, V. (2018). A Design Pattern Approach to Cartography with Big Geospatial Data. *Cartographic Journal*, *54*(4), 301–312. https://doi.org/10.1080/00087041.2017.1400199

Conallen, J. (1999). Modeling web application architectures with UML. *Communications of the ACM*, *42*(10), 63–70. https://doi.org/10.1145/317665.317677

Conceptueel model. (n.d.). Catalogus BAG 2018. Retrieved August 8, 2021, from https://imbag.github.io/catalogus/hoofdstukken/conceptueelmodel

ESRI Shapefile Technical Description. (1998). www.esri.com,

Gamma, E., Helm, E., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*.

GIS file formats - Wikipedia. (n.d.). Retrieved May 4, 2021, from https://en.wikipedia.org/wiki/GIS_file_formats

GitHub. (2020). *microsoft/TypeScript: TypeScript is a superset of JavaScript that compiles to clean JavaScript output*. GitHub. https://github.com/microsoft/TypeScript

Google Maps - Wikipedia. (n.d.). Retrieved July 6, 2021, from https://en.wikipedia.org/wiki/Google_Maps

Gordillo, S., Balaguer, F., Mostaccio, C., & Das Neves, F. (1999). Developing GIS applications with objects: A design patterns approach. *GeoInformatica*, *3*(1), 32. https://doi.org/10.1023/A:1009809511770

Gu, Q., & Van Vliet, H. (2009). SOA decision making - What do we need to know. *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK 2009*, 25–32. https://doi.org/10.1109/SHARK.2009.5069112

Home | Beeldmateriaal Nederland. (n.d.). Retrieved August 8, 2021, from https://www.beeldmateriaal.nl/

IEEE-SA Standards Board. (2000). IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Std*, *1471–2000*, 1–23. https://ieeexplore.ieee.org/document/875998

Ingram, P. (1995). The World Wide Web. *Computers and Geosciences*, *21*(6), 799–816. https://doi.org/10.1016/0098-3004(95)00012-W

Introductie - PDOK. (n.d.-a). Retrieved July 13, 2021, from https://www.pdok.nl/introductie/-/article/luchtfoto-pdok

Introductie - PDOK. (n.d.-b). Retrieved August 8, 2021, from https://www.pdok.nl/introductie/-/article/basisregistratie-topografie-achtergrondkaarten-brt-a-

ITC building - Google Maps. (n.d.). Retrieved July 6, 2021, from https://www.google.com/maps/place/ITC+building/@52.2237004,6.8837057,17z/data=!3m1!4b1!4m5!3m4!1s0x47b8138ac43f46d7:0x131179779647f980!8m2!3d52.2235317!4d6.886283?authuser=1

JPEG - Wikipedia. (n.d.). Retrieved May 5, 2021, from https://en.wikipedia.org/wiki/JPEG

Kadaster Data Science. (n.d.). Retrieved July 14, 2021, from https://data.labs.kadaster.nl/

Kadaster Labs. (n.d.). Retrieved August 7, 2021, from https://labs.kadaster.nl/cases/selfservice_eng

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object- oriented Analysis and Design and the Unified Process. In *Development* (Vol. 17). Pearson. http://www.ncbi.nlm.nih.gov/pubmed/16876023

LDWizard. (n.d.). Retrieved May 15, 2021, from https://labs.kadaster.nl/demonstrators/geodatawizard/#1

Luqun, L., Jian, L., & Yu, T. (2002). Web GIS: Technologies and Its Applications. *Dbms*, 1–6.

Maso, J., Pomakis, K., & Julia, N. (2010). OpenGIS Web Map Tile Service Implementation Standard. *Open Geospatial Consortium Inc*, 4–6. http://www.opengeospatial.org/standards/wmts

Observer pattern - Wikiwand. (n.d.). Retrieved August 15, 2021, from https://www.wikiwand.com/en/Observer_pattern

OGC Standards | OGC. (n.d.). Retrieved May 4, 2021, from https://www.ogc.org/docs/is

PostGIS — Spatial and Geographic Objects for PostgreSQL. (n.d.). Retrieved October 22, 2020, from https://postgis.net/

React – A JavaScript library for building user interfaces. (n.d.). Retrieved August 9, 2021, from https://reactjs.org/

Refactoring and Design Patterns. (n.d.). Retrieved July 31, 2021, from https://refactoring.guru/

Rowland, A., Folmer, E., & Beek, W. (2020). Towards Self-Service GIS—Combining the Best of the Semantic Web and Web GIS. *ISPRS International Journal of Geo-Information*, *9*(12), 753. https://doi.org/10.3390/ijgi9120753

Shalloway, A., & Trott, J. R. (2002). Design patterns explained: a new perspective on object-oriented design. In *AddisonWesley Publ Co*. http://www.amazon.com/Design-Patterns-Explained-Perspective-Object-Oriented/dp/0201715945

Shaw, M., & Clements, P. (1997). Field guide to boxology: Preliminary classification of architectural styles for software systems. *Proceedings - IEEE Computer Society's International Computer Software and Applications Conference*, 6–13. https://doi.org/10.1109/cmpsac.1997.624691

Simon, R., & Fröhlich, P. (2007). A mobile application framework for the geospatial web. *16th International World Wide Web Conference, WWW2007*, 381–390. https://doi.org/10.1145/1242572.1242624

Su, Y., Slottow, J., & Mozes, A. (2000). Distributing proprietary geographic data on the World Wide Web - UCLA GIS database and map server. *Computers and Geosciences*, *26*(7), 741–749. https://doi.org/10.1016/S0098-3004(99)00130-2

The Data Tier Of Your Web Mapping Architecture | GEOG 585: Web Mapping. (n.d.). Retrieved May 4, 2021, from https://www.e-education.psu.edu/geog585/node/690

Tolochko, R. C. (2016). *Contemporary Professional Practices in Interactive Web Map Design*. https://tolomaps.github.io/assets/Tolochko_Thesis.pdf

Toponamenzoeker. (n.d.). Retrieved May 26, 2021, from https://labs.kadaster.nl/demonstrators/namen-app/#/

Trygve/MVC. (n.d.). Retrieved August 2, 2021, from https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html

Vaccari, L., Shvaiko, P., & Marchese, M. (2009). A geo-service semantic integration in Spatial Data Infrastructures. *International Journal of Spatial Data Infrastructures Research*, *4*(4), 24–51. https://doi.org/10.2902/1725-0463.2009.04.art2

What is SPARQL? | Ontotext Fundamentals Series. (n.d.). Retrieved August 8, 2021, from https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/