CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF CIVIL ENGINEERING
STUDY PROGRAM: GEODESY AND CARTOGRAPHY
BRANCH OF STUDY: GEODESY, CARTOGRAPHY AND GEOINFORMATICS

BACHELOR'S THESIS

# DATABASE OUTPUT STORAGE SUPPORT IN PYWPS FRAMEWORK

Supervisor: Ing. Martin Landa, Ph.D.
Department of Geomatics

February 2018

Jan PIŠL

**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**
**Fakulta stavební**
Thákurova 7, 166 29 Praha 6

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Pišl    Jméno: Jan    Osobní číslo: 441050

Zadávající katedra: Katedra geomatiky

Studijní program: Geodézie a kartografie

Studijní obor: Geodézie, kartografie a geoinformatika

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce: Možnosti integrace databázového úložiště v rámci frameworku PyWPS

Název bakalářské práce anglicky: Database Output Storage Support in PyWPS Framework

Pokyny pro vypracování:
Bakalářská práce se věnuje návrhu rozšíření frameworku PyWPS. PyWPS je jedna z implementací standardu OGC Web Processing Service (WPS) v programovacím jazyku Python. Hlavní náplní práce je ověření možnosti integrace databázového úložiště pro výstup jednotlivých procesů. V současnosti procesy ukládají výstupní data do souborových formátů uložených přímo na výpočetním serveru, ze kterého si je poté klient může stáhnout. Integrace vzdáleného databázového úložiště by přesun výstupních dat směrem ke klientovi mohla zefektivnit. V rámci bakalářské práce se počítá s návrhem implementace zvoleného scénáře.

Seznam doporučené literatury:
Pilgrim, M.: Dive Into Python, Createspace Independent Pub 2009, ISBN: 9781441413024
OGC® WPS 2.0 Interface Standard
Obe, R.: PostGIS in Action, ISBN: 978-1935182269

Jméno vedoucího bakalářské práce: Ing. Martin Landa, Ph.D.

Datum zadání bakalářské práce: 9.10.2017    Termín odevzdání bakalářské práce: 14.1.2018
*Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku*

[redacted]
Podpis vedoucího práce

[redacted]
Podpis vedoucího katedry

## III. PŘEVZETÍ ZADÁNÍ

*Beru na vědomí, že jsem povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v bakalářské práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.*

[redacted]
Datum převzetí zadání

[redacted]
Podpis studenta(ky)

## ABSTRACT

The aim of this bachelor thesis is to design an extension for the PyWPS framework that would enable output data derived from PyWPS processes to be stored in a remote database. PyWPS is an implementation of the OGC Web Processing Service standard. Currently, output data is saved in a standard file format on the server from which the client can download it. Integration of a database output storage can make more effective both transfering data to the client and its further processing and analysis. Like PyWPS, the extension is written in Python. As for the database management system, PostgreSQL and PostGIS were used. PostGIS is an extension that adds support for geographic objects to PostgresSQL. The problem of implementing this extension within the PyWPS source code is, to some extent, also adressed in this thesis.

## KEYWORDS

PyWPS, databases, Python, GDAL, PostGIS

## ABSTRAKT

Cílem této bakalářské práce je navrhnout rozšíření frameworku PyWPS, jenž by umožnilo využít vzdálené databázové úložiště pro ukládání výstupů jednotlivých procesů. PyWPS je implementace standardu OGC Web Processing Service. Výstupní data jsou momentálně ukládána v souborových formátech na výpočetním serveru, odkud si je klient může stáhnout. Integrace databázového úložiště může zefektivnit přesun výstupních dat ke klientovi i jejich následnou správu. Rozšíření je - stejně jako framework PyWPS - napsáno v programovacím jazyce Python. Jako vhodný databázový systém byl zvolen PostgreSQL, respektive jeho nadstavba PostGIS, která přidává podporu pro geografické objekty. Součástí práce je i předběžný návrh implementace tohoto rozšíření do zdrojového kódu PyWPS.

## KLÍČOVÁ SLOVA

PyWPS, databáze, Python, GDAL, PostGIS

## STATUTORY DECLARATION

I hereby declare that this bachelor thesis is completely my own work and that I used only the cited sources in accordance with the Methodical instruction about observance of ethical principles of preparation of university final projects.

In Prague ............... ...................................

(Signature)

## ACKNOWLEDGEMENT

I would like to thank my supervisor, Ing. Martin Landa, PhD., for his guidance, advice and patience.

# Contents

# List of Figures

# 1 Introduction

Cloud computing - the delivery of on-demand computing resources over a network (internet or intranet) - is on the rise. Computing resources may range from relatively simple ones such as text editors to very complex software. They all share the same basic concept – the client only needs to provide input data, the process itself runs on a server and then returns the corresponding output data to the client.

Using cloud computing offers a way to gain new capabilities without investing in new hardware or software. It allows users to perform complex tasks without having to deal with technical details of the process. It ensures better collaboration by allowing remote teams to use the same software. And as all resources are maintained by the service provider, usually the latest version of the software is available.

For the above mentioned and other reasons, cloud computing is frequently used in the field of geoinformation technology and GIS. It is most useful when it is clear to the client what they need to do and how to do it and they only lack the tools to perform the task. For example, the ESRI's ArcGIS Online platform offers a wide selection of tools to choose from. These can be "rented" by the client and the task is then performed on the server. That way, the client gets the desired data without having to purchase (and learn to use) any desktop GIS application.

While the benefits of cloud computing are apparent, there is also another aspect of it that may not get as much attention – the management of the data produced by such processes. The amount of data collected and used is increasing rapidly and so is the number of processes that are being shared on a network. Naturally, the amount of data derived from such services is growing, too.

Since one of the primary motivation behind cloud computing is to make the service easy to use for the client, managing output data should also be carried out by the provider of the service in such a way that is convenient for the client – it needs to be effective, well-organized and easy to access.

Relational databases are one way to do so. They are designed and organized especially for rapid search and retrieval in what can be a large amount of data. Database-management system (DBMS) is a tool for managing and interacting with databases.

DBMSs have several major advantages over the traditional system where data is stored in files. Unlike the file management system, there can be more users accessing the same data concurrently without corrupting the data. Indexing speeds up the data retrieval operations. There is a standardized database language to use for queries. There are mechanisms such as data normalization that can be used to avoid duplicity of data and save storage space.

Being aware of these advantages, the main goal of this thesis is to design an extension, written in Python, that implements database storage for output data derived from geoprocessing services run within the PyWPS framework. They typically produce geographical data, so appropriate software must be used that is capable of dealing with this type of data. In this thesis, PostgreSQL is used together with its spatial database extender PostGIS.

# 2 Theory

## 2.1 Web Processing Services

The Web Processing Service (WPS) is an Open Geospatial Consortium (OGC) standard that provides rules for publishing and executing processes on the web. „The standard also defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and clients' discovery of and binding to those processes. The data required by the WPS can be delivered across a network or they can be available at the server.“ [39]

WPS uses HTTP (Hypertext Transfer Protocol) and XML (eXtensible Markup Language) for describing processes and data transfer. The first version 0.4.0 was released in 2005. Despite a new version 2.0.0 having been released in 2015, version 1.0.0 remains most widely used.

A process is essentially a function p that returns an output Y for each input X:

$$\text{p: } X \rightarrow Y$$

In case of WPS, a process is a geospatial operation, calculation or a model of any complexity. It may require one or more input arguments and always yields one or more outputs. If there are no input arguments, the process is either generating constant or random values. While WPS was designed for geospatial data, it is not restricted to them.

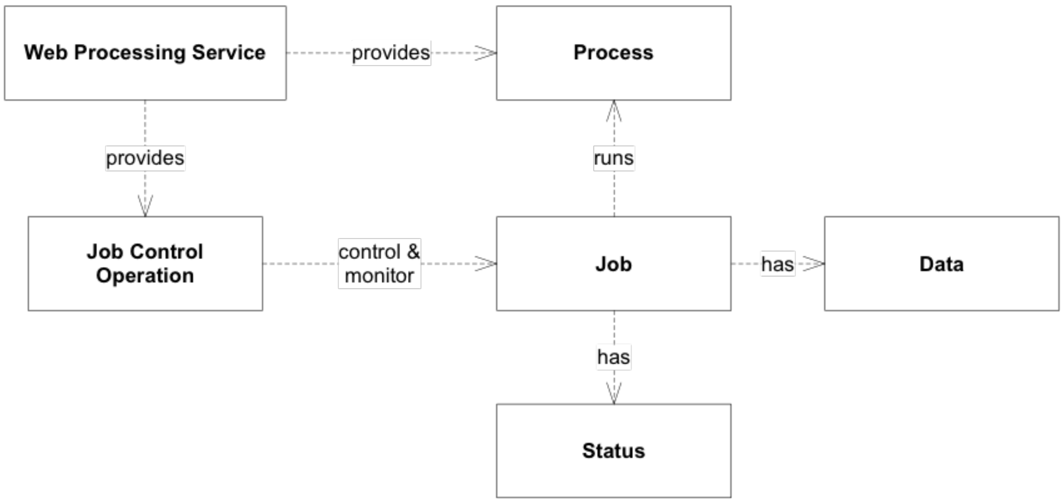

Figure 2.1: WPS conceptual model (source: Open Geospatial Consortium)

There are two basic capabilities of a WPS server. It provides the process and retrieves the process description and it controls and monitors processing jobs. Job is an instance of a process – it is an object created for a particular process execution. Job control is the ability to execute, dismiss or delete a job.

### 2.1.1 Process execution

There are two ways in which a process can be executed. If the complexity of the process is lower, the connection is stable and the completion time is relatively short (for instance, the Apache2 server uses 60 seconds as a default value for Connection-Timeout [5]) the execution is run synchronously. After the execute request is sent by the client, the WPS server starts executing the process while the client remains connected to the server for the entire time of the execution. Only when the process has finished and the output has been delivered, the connection terminates.

The asynchronous execution, on the other hand, better suits for complex processes that are expected to take longer time to finish. When the client requests process excecution, the server responds with a status information message that confirms the request has been accepted and the process will be run. The message also includes a unique processing job identifier. Then, the connection is interrupted. At this time, the client can send a GetStatus request with the job identifier to get information on how the process has progressed. Once the process has finished, the client can access the output data using a GetResult request with the job identifier. The asynchronous execution can be also useful in case of unstable connection that prevents synchronous execution to successfully run.

### 2.1.2 Available operations

**GetCapabilities.** A mandatory operation for any OGC Web service. [18] For WPS, it is one of the three basic operations that are available for any process. There are no input parameters required. Once the request is sent, the server returns a document that includes information about the service provider, a list of operations available for a WPS server and a list of all the processes offered by the service.

**DescribeProcess.** Another basic operation. The only required input is the name (or a list of names) of the process to be described. [9] If the process offers description in multiple languages, a language parameter can be added to the request. It returns a document with characteristics of the process and a description of input parameters. It also describes the output format of the process.

**Execute.** The key operation for any OGC WPS service. It allows processes that are implemented by a server to be run. [14] It requires input parameter values as specified by the service. These can be either values (numerical or other) that are included directly in the request or references to other recources that are accessible as local files or through the web. Analogically, outputs of the execute operation can be included in the XML response document or stored locally or on the web. In case of asynchronous operation, the response contains a unique JobID that the client uses to enquire about the process status and results.

**GetStatus.** An operation only used for asynchronous processes. After the Execute request has been accepted, the client can use this operation to query status information of the processing job, using a JobID returned by the Execute response. [20] This operation is only available in version 2.0.0.

**GetResult.** The final operation used for asynchronous processes that is to be used after the GetStatus operation reports the process has finished. The input parameter is the unique JobID. [19] Then, GetResult fetches the results of the process as described in the Execute section. This operation is only available in version 2.0.0.

**Dismiss.** Allows the client to let the server know the result of the process is no longer wanted. In such case, all the resources corresponding to the JobID sent in the Dismiss request may be deleted. [10] If the process is still running, it may be cancelled. This operation is only available in version 2.0.0.

### 2.1.3 OGC WPS Implementations

#### PyWPS

Figure 2.2: PyWPS (source: PyWPS)

PyWPS is a server side implementation of the OGC Web Processing Service (OGC WPS) standards 1.0.0. It is written in the Python programming language, it runs on Python 2.7, 3.3 or higher and it is tested and developed on Linux. [31] It uses a ConfigParser format for configuration files.[30] It supports a variety of geospatial software and tools such as GRASS GIS, R Project or the GDAL library. Synchronous and asynchronous invocations are supported. As for request encoding, two options are available - key-value pairs (using HTTP-GET) or XML payload (using HTTP-POST). Every process that is to be deployed on the server is defined as a class and has several mandatory parameters. The key parameter called "handler" gets invoked every time there is an incoming request, it accepts the request and returns a response.

In 2016, it upgraded from PyWPS 3 to PyWPS 4. Some of the more significant changes include every input being considered a list of inputs and all inputs having file, data and stream attributes. [29] These attributes allow better manipulation with data.

As PyWPS only supports version 1.0.0 of the OGC WPS standards, it does not support operations implemented in version 2.0.0. These operations are GetStatus, GetResult and Dismiss. Support for version 2.0.0 is currently being planned.[28]

### ZOO-Project

Figure 2.3: ZOO-Project (source: ZOO-Project)

ZOO-project is an open source WPS platform that consists of several components. The core processing engine, ZOO-Kernel, is a WPS server written in C that implements WPS standards 1.0.0 and 2.0.0.[40] A significant advantage over other WPS implementations is that it is written as a polyglot, i.e. in a valid form of multiple programming languages, which performs the same operations independent of the programming language used to compile or interpret it.[25] It runs on Mac OS, Linux and Microsoft Windows operating systems. It uses a ConfigParser styled configuration file.

ZOO-services offers a rich collection of ready-to-use services that are built on open source libraries such as GDAL or GRASS GIS.

ZOO-API is a server-side library written in JavaScript for creation and chaining services. These services can be written in one of five programming languages that are supported. It also offers easy conversion of vector formats.

ZOO-Client is a simple client-side JavaScript API for interacting with WPS from web applications. It allows to build WPS requests and send them to a WPS server. [41] It also provides functions to easily parse the output XML responses.

**52°North WPS**

Figure 2.4: 52 North (source: 52° North)

52°North WPS is a part of the 52°North open source software initiative. Located in Germany, their aim is to foster innovation in the field of geoinformatics through a collaborative process. As a part of this inititiave, the 52°North WPS is an implementation of the OGC WPS standard (version 1.0.0). It is written in the Java programming language. It can be run under Linux or Microsoft Windows operating systems.

As for the WPS invocation methods, it supports both synchronous and asynchronous invocation, HTTP-GET and HTTP-POST. As for the WPS datatypes, it supports GeoTIFF, Shapefile, KML, WKT and others. [23] Configuration is based on an XML file.

### ESRI Web Processing

Figure 2.5: ESRI (source: ESRI)

ESRI is an international company oriented on desktop and mobile GIS software, geodatabases and web GIS. Founded in 1969, it is the leading company in the global GIS market. It offers a variety of GIS products, including ArcGIS for Desktop, ArcGIS Online or ArcGIS for Mobile.

It allows services created within the ArcGIS software to be published and shared online on another ESRI's platform, ArcGIS Server. [12] On ArcGIS Server, these services can be stored and accessed by other users. They can be also implemented alongside with maps, which can also be created in other ArcGIS software and then published online, into online web applications. Users can also take advantage of the Web App Builder feature and of many templates that simplify creating applications.

By default, ESRI software does not follow WPS standards. However, when publishing a geoprocessing service in ArcGIS Desktop, there is a possibility to enable the WPS capability. Then, the service published is compliant with the OGC WPS 1.0.0 specifications.[6]

## 2.2 Spatial Databases

Spatial database (or a geodatabase) is a database optimized for storing and querying data related to objects in geometric space, such as points, lines or polygons. They require additional functionality for processing spatial data effectivelly. Usually, special data types such as geometry or feature are added along standard data types. In addition to typical SQL queries such as SELECT statements, spatial databases can perform a wide variety of spatial operations. These spatial operations include (but are not restricted to) computing line length, polygon area, distance between geometries, etc.

The International Organization for Standardization (ISO) and OGC specify a Simple Features standard that is divided into two parts. The first one defines a general model for two-dimensional geometries. [35] It also deals with spatial reference systems. The second part defines an implementation using SQL. This second part is implemented to varying extent in most of spatial databases and extensions. [35]

Here I list the best known and most widely used spatial database systems, however, there are many more. In fact, most of the major database systems support spatial data, including Microsoft SQL Server or MySQL (and its community developed branch MariaDB). Also, there is a number of database systems especially designed and developed as spatial databases (e.g. SpatialDB, SpaceBase, MapD).

### 2.2.1 PostGIS

Figure 2.6: PostGIS (source: PostGIS)

PostGIS is an open-source spatial database extension for PostgreSQL. [27] PostgreSQL is a widely-used object-relational database management system. PostGIS adds support for geographic objects according to the OGC Simple Features for SQL specification.

PostgreSQL and PostGIS use the client/server architecture. [26] When a client makes a request (typically an SQL statement), there is a server that accepts and evaluates it. The server itself is then responsible for updating (or, generally, changing in any way) the database file. Two of the most significant advantages of PostgreSQL over other DBMSs are its high standard compliance and extensibility. It allows "stored procedures" to be created and saved to simplify complex operations that are frequently repeated.

In a study that compared PostGIS with Oracle Spatial, PostGIS was found to perform faster when accessing and querying data. [1]

### 2.2.2 Oracle Spatial and Graph

ORACLE

Figure 2.7: Oracle (source: Oracle)

The Oracle Spatial and Graph is an extension of the Oracle Database that allows managing geographic data in a native type within an Oracle database. It is also a client-server service [24] but, unlike PostGIS, it is proprietary. Spatial features extend Oracle Locator, a standard feature of the Oracle Database distribution. Oracle Locator provides basic functions and services in Oracle Spatial but it lacks more advanced functions. Oracle Spatial and Graph supports large-scale geographic information systems, provides spatial web services and generally is designed for complex spatial data management and analysis.

### 2.2.3 SpatiaLite

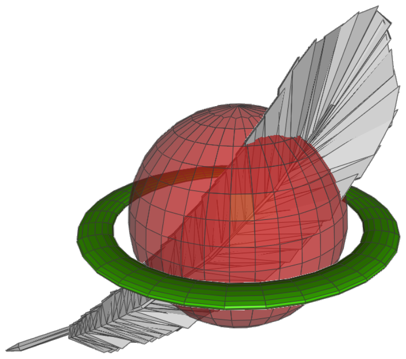

Figure 2.8: SpatialLite (source: SpatialLite)

SpatiaLite is a lightweight library that extends the SQLite database management system so it provides support for spatial data. Unlike Oracle and PostgreSQL, SQLite is designed as a single file-based database. [37] It means that, unlike server-based databases, by using SQL expressions a user updates the file directly. Therefore, a file-based database must be stored in the local file system. As a consequence, SQLite is very fast and efficient for standard operations but is not optimized for multi-user applications or for large-scale operations. SQLite (and SpatiaLite) is open-source. [38]

### 2.2.4 ArcSDE & Geodatabase (ESRI)

Figure 2.9: ArcSDE (source: ArcSDE)

ESRI is a supplier of a wide range of GIS software, including web GIS services, desktop and mobile applications and geodatabase management systems. Data used, produced or derived from ESRI software is stored in geodatabases, that are defined by ESRI as "a collection of geographic datasets of various types held in a common file system folder, a Microsoft Access database, or a multiuser relational DBMS."[11] As the definition suggests, ESRI distinguishes between three conceptually different types of geodatabases, depending on the scale of the data, requirements of the client, operating system and other factors.

In its simplest form, an ESRI geodatabase can be a collection of GIS data stored as files in a folder within the standard file system. This is called file geodatabase and is intended for single users and small workgroups. Personal geodatabase is another type of geodatabase, also designed to be used by individuals or small workgroups. It uses Microsoft Access database management system and therefore is only available on Microsoft Windows operating system.

The final and most complex type of geodatabase is the enterprise geodatabase. Unlike the previous two, it is designed to be used by multiple users simultaneously and for large datasets. It works with various DBMSs storage models using ArcSDE.

ArcSDE is a proprietary technology for managing and accessing spatial data that supports multiple DBMSs: IBM DB2, IBM Informix, Microsoft SQL Server, Oracle, and PostgreSQL.[7] It also supports the corresponding standards. ArcSDE is built with the client/server architecture. A client first sends a request to the server. Then, the server receives the request, generates results, and delivers them to the client.

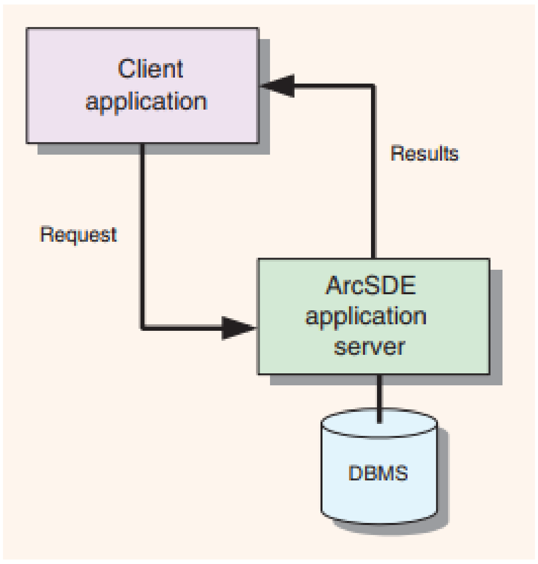

Figure 2.10: ArcSDE logic(source: ArcSDE)

The client doesn't need any knowledge of particulars of any of the **DBMS**s. Another significant advantage is that ArcSDE allows datasets to be available to multiple users for viewing, querying or editing at the same time.[13]

## 2.3 Background research

PyWPS is only one of the many implementations of the WPS standard, each of them approaching the problem of storing output data differently. From those listed above, the ZOO-project provides the "ZOO-kernel optional database support" [42] that is most similar to what the author of this thesis is aiming to create. There is an optional section in the configuration file that allows to configure the connection. The section has six elements (dbname, port, user, host, type and schema) that are used to generate a connection string that is passed to the GDAL library that connects to the database.[43]

When using ArcGIS Server (ESRI software for executing processes on a server), ArcSDE handles storing data in a database. For more information about the ArcSDE technology, refer to the section 2.2.4 - ArcSDE & Geodatabase (ESRI).

As for the 52°North Web Processing Service, it, by default, stores output data as web accessible resources and provides the consumer with an URL. Depending on the type of the output data, it may be stored directly as WMS, WFS or WCS layers. [23]

As an alternative, it is also possible to store the data within a PostgreSQL database. [4] It is, however, primarily used to save requests and responses, e.g. to help debugging the service. While storing output data in the database is technically possible, too, no spatial database is implemented at the moment.[1]

[1]Information obtained from personal email correspondence with Benjamin Proß, the contact person for 52°North WPS

# 3 Technology

## 3.1 Python

Figure 3.1: Python logo (source: Python.org)

Python is a high-level programming language that fully supports object-oriented and structured programming. Developed in the late 1980s, the first version 0.9.0 was released in 1991. In 2008, Python 3.0 was released. Currently, the most up-to-date version available is 3.6.[2]

It was designed as a syntactically simple language, using whitespace intendantion instead of brackets and English words rather than punctuation. It is a dynamically-typed language, which means it is not neccessary to specify a data-type when defining a variable. For its simplicity and readability, Python is often considered a good first programming language to learn.

One of the key advantages of Python is its high extensibility. It provides large standard libraries and also an extensive number of other modules, packages and libraries, so most of the common programming tasks are already solved, scripted and made available.

## 3.2 GitHub

Figure 3.2: GitHub logo (source: GitHub.com)

GitHub is a web-based Git repository hosting service with a graphical interface. Git is an open-source version control system for tracking changes in text files, typically used for source code management.[21] On top of the standard Git functionality, GitHub provides a number of its own features, including forking (copying a repository), pull requests, or bug tracking. GitHub also offers a desktop application.

## 3.3 Geospatial Data Abstraction Library

Figure 3.3: GDAL logo (source:[15])

Geospatial Data Abstraction Library (GDAL) is the most widely used data acces library for raster and vector geospatial data formats. It is released under an X/MIT style Open Source license by the Open Source Geospatial Foundation and it is written in C++ and C programming languages. As for operating systems, it can run under Linux, Solaris, Mac OS X and Microsoft Windows.[15]

The first version was released by Frank Warmerdam in 2000 and the last stable version 2.2.3 was released in November 2017.[17]

The OGR library was developed separately but is now a part of the GDAL source tree. GDAL used to work with raster data and OGR with vector data. Starting with GDAL 2.0, however, the two have been integrated more tightly.

For its extensive capabilities and comprehensive set of functionalities, the GDAL library is widely used by both commercial and non-commercial GIS projects and programs. The list of software programs that uses it includes Google Earth, ArcGIS, GRASS GIS and many others.[16]

## 3.4 PostGIS

For information on the PostgreSQL spatial database extender, refer to the section 2.2.1.

# 4 Implementation

## 4.1 Functionality

PyWPS allows to publish and consume geoprocessing services on a server. Every process that is to be implemented by PyWPS must be constructed as a Python class and contain a list of inputs and outputs. Also, there must be a handler method with two parameters - request and response. [33] Details on the procedure of creating new processes can be found in PyWPS documentation.

To send a request to PyWPS, an instance of PyWPS must be running at a server. The request is handled and a response is generated and returned to the client. The response has a form of an XML document that contains different elements depending on the type of the request.

When an Execute request is called, a new, temporary folder is created in location specified in configuration file and input data is copied here. While the process is being executed, temporary files may be generated in this folder. For every process, it must be specified what the final output is. Once the execution is finished, the output is copied to a location that is accessible via the web. The temporary folder, containing input and output data and all the intermediate data that arose during the execution, is then deleted.

### 4.1.1 Output Data Management

#### Current Options

The simplest option of delivering output data is to embed it in the XML response document. Either as plain text, GML or, in case of an image, base64 encoding scheme. This is typically used when the output is relatively small. For PyWPS, it is also the default option.

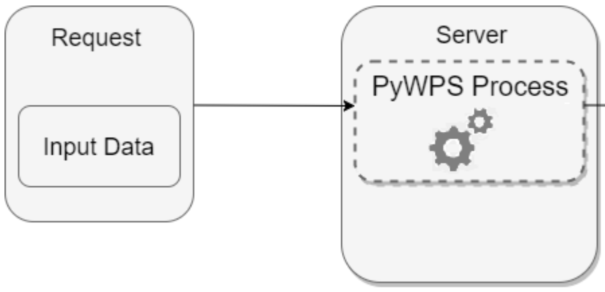

Figure 4.1: Delivering output data directly to the client (source: author)

If, on the contrary, the output data is large and complex, there is another option. The client is only given a reference, a URL link, from which the data can be downloaded. PyWPS saves the file in a folder specified in configuration passed by the service (or in a default location). The URL is embedded in the XML response. [34]

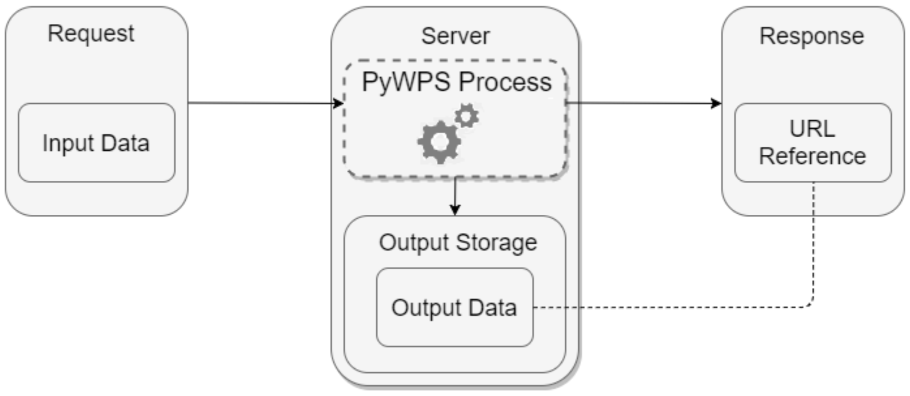

Figure 4.2: Delivering output data via URL reference (source: author)

It is up to the consumer of the process to decide which option to choose. For the latter option, the "`@asReference`" value must be set to "`True`" in the request. [8] By default, it is set to "`False`".

### Proposed Extension

The aim of this thesis was to develop another variant to add to the existing two that stores output data in a PostGIS database.

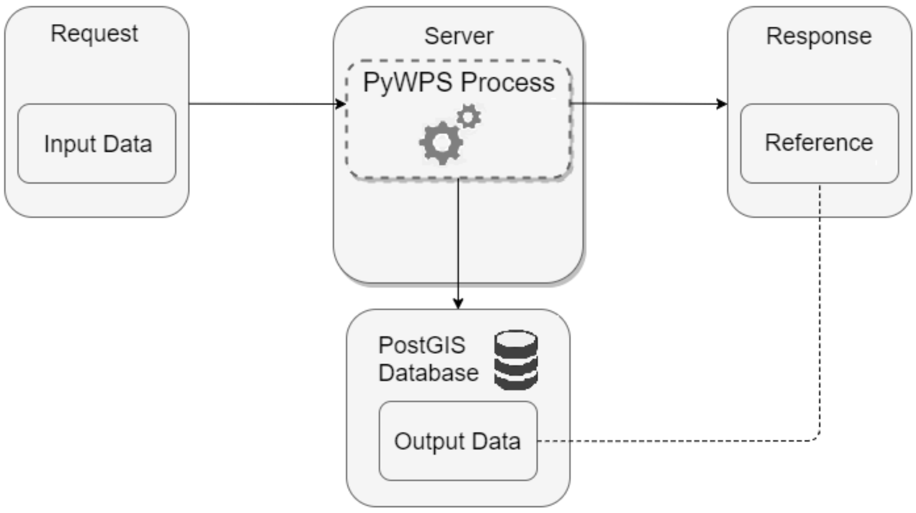

Figure 4.3: Storing output data in a remote database (source: author)

From the point of view of the consumer of a process, it is similar to the previous option. After the final output has been produced, connection to the database is established and the output data is copied there. When the XML response is delivered to the client, it contains a reference that points to the location of the data within the database. The reference is composed of the name of the database, schema and table. To access the data itself, database login credentials are neccessary.

The current state is not definitive. In the final state, the client will receive a URL link that references to a WFS service. The WFS service itself will access the output data stored in a database and serve them to the client.

To use this functionality as a consumer of a process, it must be implemented in the process by its author. For details on implementation when creating a process, please refer to appendix B.

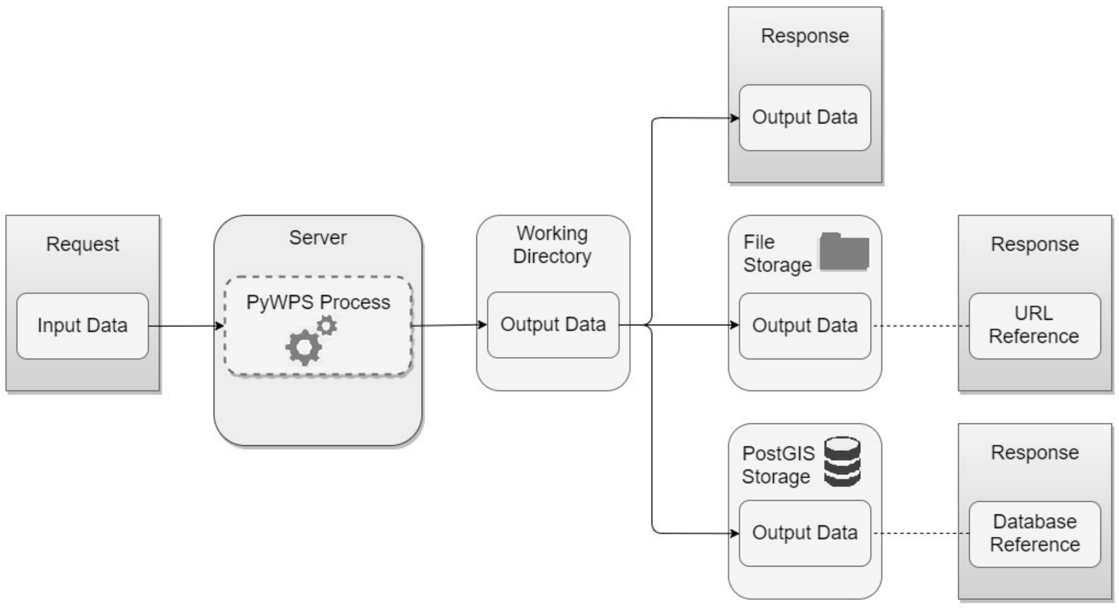

Figure 4.4: Three options of delivering output data to the client (source: author)

## 4.2 Development

### 4.2.1 PgStorage class development

A new class, `PgStorage`, has been developed that implements PostGIS storage support. Details on PostGIS can be found in section 2.2.1. `PgStorage` is a derived class, inheriting from `StorageAbstract` class that is part of PyWPS API.

PgStorage is stored within `storage.py` in the pywps.inout module. It consists of several methods that are described below.

**__init__**

A constructor, i.e. it gets called automatically when an instance of the `PgStorage` class is created.

In this method, the `get_config_value` function is used that is defined within the PyWPS API. It accesses the configuration file and retrieves required elements. What elements are retrieved is specified by the function's two parameters, section and option.

Correct section (`db`) is specified and saved to a variable. The name of the database is extracted from the configuration file and saved to a variable.

Another variable is defined that serves as a connection string for connecting to a database. Requisite elements (user name, password and host server) are retrieved from the configuration file. Finally, an instance of the `_create_schema` method is created and saved to a variable. See an example of a `db` section in section B.1.

#### _create_schema

First defines a variable `schema_name` as a random string of specified length that consists of letters and digits. This is done using Python libraries `random` and `string`.

`Psycopg2` library is used to connect to database specified by the target variable. A try-except clause is included to raise an exception if the connection cannot not be established.

Then, when a cursor has been created, an SQL query is executed that generates a new schema if it doesn't exist already. Changes in database are commited so they persist after connection is aborted, cursor and connection are closed and the `schema_name` variable is returned.

#### _store_output

As its name suggests, it handles writing output data to the database. It benefits from an extensive use of GDAL library. More information about GDAL can be found in section 3.3. It has two input parameters, the name of the file that is to be stored in a database, and a process identifier.

Thanks to GDAL, the process is fairly simple and straight-forward. The output file is opened using the file name input parameter and connection to the database is established. Then, data is copied from the output file to the database using the OGR `CopyLayer` function.

Each of the three above mentioned operations is followed by a simple condition that checks if the variable storing output of the operation is not None. If it is, it raises an exception with a corresponding message.

This method returns the identifier.

#### store

The store method is defined in the `StorageAbstract` parent class. Just as in the parent class, it has `output` as an input argument. In this case, output is an instance of the `ComplexOutput` class.

It initializies the `_store_output` method and passes it name and identifier of the output file.

Then, a string is created that specifies the location of the data and saved as a variable. It consists of a name of the database, schema and identifier. This string is then given to the client as an output in the XML response of the process.

There are three parameters returned by the method - the corresponding value of a DB variable defined within the `STORE_TYPE` class, name of the output file and the variable describing the location of the data that is composed of names of the database, schema and table. These must be returned as they are required by the `get_url` method (defined within the `ComplexOutput` class).

### 4.2.2 PyWPS source code changes

All changes that have been done within the PyWPS source code can be examined in a `diff` folder that is appended to this thesis.

#### Outputs module

This module includes three classes that define how outputs are handled and delivered to the client. Each class deals with one type of output data - Literal, Complex and BoundingBox. Since there is another option being added of storing output data that returns a reference to the client, `_execute_xml_reference` method in `ComplexOutput` class had to be adjusted.

Whether the output data is stored as a file or in a database depends on value of the `store_type` option in configuration file. `store_type` is a new variable within the configuration file that was declared for this purpose. For details on changes in the configuration file, refer to section B.1. The code block that has been added here first retrieves the value using `get_config_value`. If it is equal to `"db"`, `PgStorage` is chosen to handle output data. In any other case (the value is different or the option is missing), `FileStorage` is used.

#### Storage module

`PgStorage` class has been added that implements the database storage functionality. For more details on this class, refer to the section 4.2.1.

`FileStorage` is another class defined in this module that inherits from the `StorageAbstract` class. As described above, it is either this class or `PgStorage` that gets called when a reference is to be returned within the response document. As its name indicates, it saves outputs as files.

There is another class called `get_free_space` within this module. Its name, too, is self-explanatory - it returns folder or drive free space.

Also, second option has been added to the class `STORE_TYPE`. So, apart from a `PATH` variable that implies storing output data as a file, there is also a `DB` variable that is used when saving data in a database.

### 4.2.3 Testing

A script has been developed to test whether the process of storing outputs in a database functions correctly.

For the purpose of this test, three simple processes have been written. One of them only returns a string, while the other two, (`process_one_output` and `process_two_outputs`), produce one and two complex outputs, respectively.

Both `process_one_output` and `process_two_outputs` generate buffers around input features, the latter also calculates centroids thereof. They are based on sample processes available for the PyWPS demo service. There is also a GML file provided with the demo service that was used as an input file for this test.

To sucessfully run the test, instance of PyWPS must be running. When run, the test executes each of the processes and analyzes the corresponding XML response using the ElementTree XML parser. For every process, it returns an identifier of the process extracted from the XML document.

For the two processes that yield complex outputs the test establishes connection to the database and counts features in the corresponding table. Then it compares this value to the number of features in the input file. If these two values differ, it raises an exception.

Similarly, it checks whether the geometry type of the layer in database is equal to a predefined value (point for centroids, polygon for buffer). If not, it raises an exception with a corresponding warning.

When no exception is raised, it indicates that all processes have been run and all complex outputs have been stored in a database.

GDAL library is used for creating database connection, counting features and getting geometry type. Database login credentials are retrieved from a configuration file using `get_config_value`. To ensure correct configuration file is read, another PyWPS built-in function, `load_configuration`, is used. For information on how to download all required data and run the test, refer to appendix A.

# 5 Conclusion and future work

The aim of this thesis was to design an extension for the PyWPS framework that would allow output data to be stored in a database rather than in a standard file system.

Until now, there were only two ways of returning data to the client. It was either embedded in the response directly or, typically if the data was larger or more complex, it remained stored on a server and the response included a reference (a URL link) pointing to the location from which the data could be downloaded.

By adding the third option of storing outputs in a remote database, the output data can be transferred, stored and processed more effectively. However, there is a lot of room for improvement and future work.

Most importantly, current state of the extension provides the client with a string that points to a specific table, schema and database. The client, however, has to access the database by themselves and only then work with the data.

In the future, the client should be only given a unique URL link that points to a running WFS (or, for viewing data only, WMS and WCS) service that will be retrieving data from the database. The client will not even need to be aware where tha data is stored as they will access it easily through a standardized interface.

The author encountered several other issues during the development. Some of them have been solved, others were out of the scope of this thesis.

One of the unresolved issues is safety of the database login credentials. At this point, all the data that is neccessary for connecting to and accessing the database (including password) is stored in the configuration file as plain text. Obviously, this is a major safety risk and a better, more secure solution is needed where (at least) the password would not be accessible directly.

Another problem that may arise if using this extension and that should be addressed in the future is exceeding the capacity of the database if the output data that is being copied to the database is too large. This could be solved by adding another functionality that would establish a connection to the database, check how much space there is available and then compare it to the size of the output file and raise an exception if the capacity was not sufficient.

These and perhaps other improvements are neccessary before the extension can be fully implemented by PyWPS. Due to time constraints, the above mentioned changes will be worked on after this thesis has been submitted. The author plans to cooperate with authors of PyWPS to implement the extension as a pull request to the PyWPS repository. The entire thesis, including text, source code and sample data, is available on GitHub at: https://github.com/ctu-geoforall-lab-projects/bp-pisl-2018.

# List of Acronyms

| | |
|---|---|
| WPS | Web Processing Service |
| OGC | Open Geospatial Consortium |
| HTTP | Hypertext Transfer Protocol |
| XML | Extensible Markup Language |
| DBMS | Database Management System |
| API | Application Programming Interface |
| TIFF | Tagged Image File Format |
| KML | Keyhole Markup Language |
| WKT | Well-known text |
| GDAL | Geospatial Data Abstraction Library |
| GIS | Geographic Information System |
| SQL | Structured Query Language |
| RDBMS | Relational Database Management System |
| WMS | Web Map Service |
| WFS | Web Feature Service |
| WCS | Web Coverage Service |
| OSGeo | Open Source Geospatial Foundation |
| ISO | International Organization for Standardization |
| GML | Geography Markup Language |
| URL | Uniform Resource Locator |

# Bibliography

[1] DEEPIKA SHUKLA, Darshit Shah C. S. *PostGIS and Oracle Spatial Performance Comparison* [online]. [cit. 2018-01-04]. Available at: `http://csjournals.com/IJCSC/PDF7-2/16.%20Deepika.pdf`.

[2] PILGRIM, Mark. *Dive Into Python.* Apex: Apress, 2004. ISBN 978-1-59059-356-1.

[3] PROß, Benjamin. Private email communication.

[4] *How to use PostgreSQL for request/response/result storage* [online]. [cit. 2018-01-11]. Available at: `https://wiki.52north.org/Geoprocessing/WPSAndPostgreSQL`.

[5] *Apache2 Httpd Server Connection Timeout Default Value* [online]. [cit. 2018-01-02]. Available at: `http://httCopd.apache.org/docs/2.4/mod/core.html#timeout`.

[6] *ESRI WPS Compliance* [online]. [cit. 2018-01-06]. Available at: `http://server.arcgis.com/en/server/latest/publish-services/linux/wps-services.htm`.

[7] *Understanding ArcSDE* [online]. [cit. 2018-01-06]. Available at: `http://downloads.esri.com/support/documentation/sde_/706Understanding_ArcSDE.pdf`.

[8] *Returning Large Data* [online]. [cit. 2018-01-12]. Available at: `http://pywps.readthedocs.io/en/master/process.html#returning-large-data`.

[9] *DescribeProcess Request* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#55`.

[10] *Dismiss Extension* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#86`.

[11] *ESRI Geodatabase Definition* [online]. [cit. 2018-01-06]. Available at: `http://desktop.arcgis.com/en/arcmap/10.3/manage-data/geodatabases/what-is-a-geodatabase.htm`.

[12] *How to publish a service from ArcMap or ArcCatalog* [online]. [cit. 2018-01-11]. Available at: `http://enterprise.arcgis.com/en/server/latest/publish-services/linux/how-to-publish-a-service.htm`.

[13] *ESRI Geodatabase Types* [online]. [cit. 2018-01-06]. Available at: `http://desktop.arcgis.com/en/arcmap/10.3/manage-data/geodatabases/types-of-geodatabases.htm`.

[14] *Execute Operation* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#58`.

[15] *GDAL - Geospatial Data Abstraction Library* [online]. [cit. 2018-01-6]. Available at: `http://www.gdal.org/`.

[16] *GDAL/OGR Info Sheet* [online]. [cit. 2018-01-07]. Available at: `http://www.osgeo.org/gdal_ogr`.

[17] *GDAL/OGR 2.2.3 Release Notes)* [online]. [cit. 2018-01-07]. Available at: `https://trac.osgeo.org/gdal/wiki/Release/2.2.3-News`.

[18] *GetCapabilities Operation* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#50`.

[19] *GetResult Request* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#67`.

[20] *GetStatus Operation* [online]. [cit. 2018-01-11]. Available at: `http://docs.opengeospatial.org/is/14-065/14-065.html#62`.

[21] *Git* [online]. [cit. 2018-01-06]. Available at: `https://en.wikipedia.org/wiki/Git`.

[22] *Python hassatr Function* [online]. [cit. 2018-01-07]. Available at: `https://docs.python.org/3.6/library/functions.html#hasattr`.

[23] *52 North Web Processing Service features* [online]. [cit. 2018-01-06]. Available at: `https://52north.org/software/software-projects/wps/`.

[24] *Oracle7 Server Concepts Manual* [online]. [cit. 2018-01-11]. Available at: https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch20.htm.

[25] *Polyglot (computing)* [online]. [cit. 2018-01-06]. Available at: https://en.wikipedia.org/wiki/Polyglot_(computing).

[26] *Architectural Fundamentals* [online]. [cit. 2018-01-11]. Available at: https://www.postgresql.org/docs/9.1/static/tutorial-arch.html.

[27] *About PostGIS* [online]. [cit. 2018-01-11]. Available at: https://postgis.net.

[28] *PyWPS Documentation* [online]. [cit. 2018-01-02]. Available at: https://pywps.readthedocs.io/en/master/.

[29] *Single process definition* [online]. [cit. 2018-01-11]. Available at: http://pywps.readthedocs.io/en/master/migration.html#single-process-definition.

[30] *Configuration* [online]. [cit. 2018-01-11]. Available at: https://pywps.readthedocs.io/en/master/configuration.html#configuration.

[31] *Dependencies and requirements* [online]. [cit. 2018-01-11]. Available at: http://pywps.readthedocs.io/en/master/install.html#dependencies-and-requirements.

[32] *PyWPS Processes* [online]. [cit. 2018-01-07]. Available at: http://pywps.readthedocs.io/en/master/process.html.

[33] *PyWPS Processes* [online]. [cit. 2018-01-07]. Available at: http://pywps.readthedocs.io/en/master/process.html.

[34] *PyWPS Processes - Returning Large Data* [online]. [cit. 2018-01-07]. Available at: http://pywps.readthedocs.io/en/master/process.html#returning-large-data.

[35] *Simple Feature Access - Part 1: Common Architecture* [online]. [cit. 2018-01-11]. Available at: http://www.opengeospatial.org/standards/sfa.

[36] *Simple Feature Access - Part 2: SQL Option* [online]. [cit. 2018-01-11]. Available at: http://www.opengeospatial.org/standards/sfs.

[37] *Single-file Cross-platform Database* [online]. [cit. 2018-01-11]. Available at: https://www.sqlite.org/onefile.html.

[38] *SpatiaLite* [online]. [cit. 2018-01-12]. Available at: https://www.gaia-gis.it/fossil/libspatialite/index.

[39] *Open Geospatial - WPS standard* [online]. [cit. 2018-01-04]. Available at: http://www.opengeospatial.org/standards/wps.

[40] *ZOO-Kernel Compliance* [online]. [cit. 2018-01-04]. Available at: http://www.zoo-project.org/docs/kernel/what.html#compliant.

[41] *What is ZOO-Client?* [online]. [cit. 2018-01-11]. Available at: http://www.zoo-project.org/docs/client/what.html#what-is-zoo-client.

[42] *ZOO-project Database Backend* [online]. [cit. 2018-01-06]. Available at: http://www.zoo-project.org/docs/install/installation.html#zoo-create-db-backend.

[43] *ZOO-project Database Section* [online]. [cit. 2018-01-06]. Available at: http://www.zoo-project.org/docs/kernel/configuration.html#database-section.

# Appendix

# A User guide for testing

I. Clone the repository:

```
$ git clone \
https://github.com/ctu-geoforall-lab-projects/bp-pisl-2018
```

II. Using requirements.txt, go to bp-pisl-2018/src directory and install all required packages, including PyWPS core package:

```
$ pip3 install -r requirements.txt
```

III. Edit configuration file (refer to appendix B for details)
IV. Run demo application (taken from PyWPS-Demo):

```
$ python3 demo.py
```

V. Run the test in another terminal - demo application must be running concurrently:

```
$ python3 test.py
```

# B Enabling database storage

To enable the database storage capacity as an author of a process, there are a few things that must be done. No changes are neccessary in the code of the process itself, but configuration file must be updated.

It is assumed that there is an instance of PostGIS database running on some server.

## B.1 Configuration file changes

When output data is not embedded directly in the response document, there are two more options - it can be stored as a file or in a database. The decision is made based on a value of the `store_type` variable in the configuration file. Therefore, a new option of this name must be added in the `server` section and its value set to `db`. If the option already exists, only the value must be changed. Position of the option within the `server` section is arbitrary.

```
store_type = db
```

To connect to a remote database, login credentials are required such as the database name, user name, password and a host (server the database runs on). The `PgStorage` class is designed to extract them from the configuration file, so the author must add a new section there that contains all the required information. An example of a complete `db` section is shown below.

```
[db]
host=geo102.fsv.cvut.cz
user=pisl
password=XXXXXXXX
dbname=pisl_bp
```

# C GitHub Repository Content

The repository is available on GitHub at: https://github.com/ctu-geoforall-lab-projects/bp-pisl-2018.

```
repository
├── src
│   ├── diff
│   ├── processes
│   └── static
│       └── data
├── text
│   ├── pictures
│   └── text of the thesis as a PDF file
└── zadani
```