SandDB - Community Sand Database

Vasco Guilherme Brás Pinho

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. João Nuno de Oliveira e Silva
Prof. Anabela Gonçalves Cruces

Examination Committee

Chairperson: Prof. Nuno Cavaco Gomes Horta
Supervisor: Prof. João Nuno de Oliveira e Silva
Member of the Committee: Prof. Rui Pires de Matos Taborda

November 2015
Acknowledgments

Thanks Mom and Dad. Thanks Rita. Thanks Friends.
Obrigado também aos meus avós.
Resumo

Mundialmente, são recolhidas todos os anos inúmeras amostras geológicas que são habitualmente armazenadas de formas que dificultam a descoberta e colaboração na investigação. Nesta tese apresenta-se uma plataforma online que permite a investigadores, professores, alunos e entusiastas da geologia, colaborar, armazenar, partilhar e analisar dados sedimentológicos. Implementou-se uma base de dados georeferenciada de amostras sedimentológicas acessível a utilizadores e serviços web através de uma API REST e um sítio online.

Apresentamos os métodos utilizados actualmente por geólogos para a recolha de amostras sedimentológicas, as ferramentas que existem para facilitar estes processos e os problemas que encontrámos nas mesmas. A partir deste trabalho inicial formulou-se um conjunto de especificações que deram origem à implementação da plataforma SandDB. Com esta tese implementa-se uma plataforma de base à criação de um sistema GIS completo, baseado em tecnologias web open-source. Com este sistema pretende-se que qualquer utilizador consiga de forma fácil ter acesso a ferramentas de armazenamento e partilha de dados sedimentológicos.

Keywords: Bases de dados georeferenciadas, sedimentos, amostras sedimentológicas, plataformas online, base de dados de sedimentos, API REST.
Abstract

A large number of geological samples is collected each year around the world which are usually stored in a way that prevents discoverability and collaborative research. With this thesis we present a better online tool to allow researchers, teachers, students and geology enthusiasts to store, share, analyze and collaborate on sedimentological data. We implemented a georeferenced database of sediment samples accessible by users and web services through a REST API and an online website.

Current methods used by geologists for sediment sample collection are presented, as well as which tools currently exist to facilitate these processes and the problems we found with them. A set of requirements is identified in order to improve on existing tools, and the implementation of the SandDB. With this thesis we implement the basis for full fledged GIS system based on open-source web software. Our aim with this system is to empower users with tools for easy storage and sharing of sediment data.

Keywords: Georeferenced databases, sediments, geological samples, online platforms, sediment database, REST API.
# Contents

Acknowledgments .......................................................... iii
Resumo ................................................................. iv
Abstract ................................................................. v
List of Tables ........................................................... viii
List of Figures .......................................................... x
List of Acronyms ........................................................ xi

1 Introduction ............................................................. 1
   1.1 Problem Statement .................................................. 1
   1.2 Objectives .......................................................... 2
   1.3 Document Outline .................................................. 3

2 State of the Art ......................................................... 5
   2.1 Sample collection, archival and analysis ....................... 5
      2.1.1 Collection and Archival .................................... 5
      2.1.2 Analyses .................................................... 5
   2.2 Geospatial protocols, standards and libraries ................ 8
      2.2.1 Open Geospatial Consortium ................................ 9
      2.2.2 Geospatial data representation formats .................... 9
      2.2.3 Geospatial Libraries ....................................... 11
      2.2.4 Geospatial data storage .................................... 12
      2.2.5 Protocols and methods for geospatial data processing and presentation ............................................. 14
      2.2.6 Web Mapping ............................................... 16
      2.2.7 GIS Desktop and Server Applications ....................... 17
      2.2.8 Overview of geospatial software by features .......... 18
   2.3 Web applications using GIS servers ............................ 20
      2.3.1 Governmental web applications ............................ 20
      2.3.2 Geological systems focused on sediment storage and analysis ................................................................. 21
      2.3.3 Other local tools ........................................... 22

3 Supporting technologies and tools .................................. 25
   3.1 Architecture requirements for a Geospatial Web Application ................................................................. 25
   3.2 Front-end requirements for a Geospatial Web Application ................................................................. 25

4 SandDB ................................................................. 27
   4.1 Supporting technology ............................................. 27
   4.2 Django architecture overview and request life-cycle .......... 29
   4.3 Data modeling ..................................................... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.1 Samples</td>
<td>32</td>
</tr>
<tr>
<td>4.3.2 Users and permissions</td>
<td>33</td>
</tr>
<tr>
<td>4.4 REST API design</td>
<td>36</td>
</tr>
<tr>
<td>4.4.1 Read-only, or safe methods</td>
<td>36</td>
</tr>
<tr>
<td>4.4.2 Methods with side effects</td>
<td>36</td>
</tr>
<tr>
<td>4.4.3 Overview of implemented endpoints</td>
<td>37</td>
</tr>
<tr>
<td>4.5 Django-wfs app</td>
<td>38</td>
</tr>
<tr>
<td>4.6 User Interface and User Experience</td>
<td>42</td>
</tr>
<tr>
<td>4.6.1 External dependencies</td>
<td>42</td>
</tr>
<tr>
<td>4.6.2 User registration flow</td>
<td>42</td>
</tr>
<tr>
<td>4.6.3 Importing and Exporting data</td>
<td>43</td>
</tr>
<tr>
<td>4.6.4 Other pages</td>
<td>44</td>
</tr>
<tr>
<td>4.6.5 Deployment</td>
<td>49</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>50</td>
</tr>
<tr>
<td>5.1 Future Work</td>
<td>50</td>
</tr>
<tr>
<td>Bibliography</td>
<td>55</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Overview of relevant sample data at collection time. ................................................. 6
2.2 Overview of relevant sample data at archival time. .................................................. 6
2.3 Overview of sieving data. ......................................................................................... 6
2.4 Heavy mineral separation data. ................................................................................. 7
2.5 Munsell color data. ................................................................................................... 7
2.6 Moisture content data. ............................................................................................. 7
2.7 Atterberg limits data. ................................................................................................. 8
2.8 Soluble salts/conductivity data. ............................................................................... 8
2.9 pH data. .................................................................................................................... 8
2.10 Total and organic carbon data. ............................................................................... 9
2.11 Data store feature comparisons. ............................................................................. 13
2.12 Mapping solutions for web clients. ....................................................................... 17
2.13 Comparison of MapServer and GeoServer features. ........................................... 18
2.14 Overview of server-side geospatial open source software by features. ................. 19

4.1 Comparison of Python web frameworks ................................................................. 28
4.2 Safe methods overview for a generic resource. ...................................................... 36
4.3 Side effect methods overview for a generic resource. .......................................... 37
4.4 Implemented API endpoints. .................................................................................. 37
4.5 Description of existing WFS 1.0.0 request types. ................................................. 40
List of Figures

2.1 Graph of software and protocol relationships from OGC. .......................... 9
2.2 A raster image requested with WMS overlayed on an OpenStreet Map. .......... 14
2.3 Trimming and slicing with a WCS bounding box. .................................... 15
2.4 Both the base layer and the overlays can be loaded by tiling. ....................... 15
2.5 WFS feature overlays on an OpenStreet Map. ....................................... 16
2.6 QGIS desktop application. .............................................................. 17
2.7 GeoServer import and export formats and protocols. ................................. 18
2.8 Online government programs for geoscience. .......................................... 20
2.9 GeoNetwork OpenSource. ....................................................................... 21
2.10 Sample search User Experience in SedDB. ............................................. 22
2.11 Sample search in SedDB and SESAR. .................................................. 22
2.12 The GRADISTAT’s spreadsheet data input. ............................................. 23
2.13 GRADISTAT’s generated gravel sand mud diagram. .................................. 24

3.1 Generic geospatial enabled web application architecture. .......................... 26
3.2 Generic geospatial enabled web application front-end. ............................... 26

4.1 Django REST Framework browsable API. ............................................... 28
4.2 Chosen architecture for SandDB. ......................................................... 30
4.3 Django architecture overview. Adapted from mytardis.readthedocs.org/en/latest/architecture.html .......................................................... 31
4.4 Classifications relation with Sample. ...................................................... 33
4.5 Granulometry test models. ...................................................................... 33
4.6 PDF files for generic tests, photos and photo type models. ......................... 34
4.7 Django user model, sessions, authentication and permissions. .................... 35
4.8 How to use the GeoJSON API with QGIS. .............................................. 38
4.9 Models for WFS implementation. ............................................................ 39
4.10 Administration panel for the WFS FeatureType associated with a SandDB User. 40
4.11 How to use the WFS service with QGIS. ............................................... 41
4.12 User registration flow. .......................................................................... 42
4.13 User sign in flow. .................................................................................. 43
4.14 Bulk import of data through a custom form. ........................................... 43
4.15 Spreadsheet interface in the browser. ................................................... 44
4.16 Import from spreadsheet file flow. ......................................................... 44
4.17 SandDB user dashboard page. Test data. ............................................... 45
4.18 SandDB campaign page. Test data. ....................................................... 46
4.19 SandDB sample page. Test data. .................................................. 47
4.20 SandDB filtering samples by classification type. Test data. .................. 48
List of Acronyms

ACID  Atomicity, Consistency, Isolation, Durability
API   Application Programming Interface
BGS   British Geological Survey
CRUD  Create, Read, Update, Delete
CSS   Cascading Style Sheets
DBMS  Database Management System
GIS   Geographic Information System
HTML  HyperText Markup Language
HTTP  Hypertext Transfer Protocol
IEDA  Integrated Earth Data Applications
IGP   Portuguese Geographic Institute
IGSN  International Geo Sample Number
JSON  JavaScript Object Notation
LNEG  National Energy and Geology Laboratory
NSF   National Science Foundation
PDF   Portable Document Format
RDBMS Relational Database Management System
REST  Representational State Transfer
SESAR System for Earth Sample Registration
SSH   Secure Shell
SedDB Sediments Database
UI    User Interface
URL   Uniform Resource Locator
USGS  United States Geological Survey
WSGI  Web Server Gateway Interface
1. Introduction

In science there are problems for which solutions developed in different fields might have a sizable impact. These solutions or small incremental contributions can be hiding in plain sight, being obvious in one field and unknown in another. By developing tools which leverage known practices in one area we can usually make researchers lives much easier in other areas. The specific case of geologic research has, in our possibly uninformed view, remained largely unimproved by the possibilities of Internet based social networks. No collaborative system exists where researchers can store, share, analyze and visualize sedimentological data.

Sediments have embedded in their physical and chemical properties answers to their origins and those of their environments [6]. Many scientific questions, policy issues [9] and social problems can benefit from the answers provided by the analysis of sediments. In Geology, this study is composed of several techniques to gather information on trends in the processes that regulate transportation and deposition of sediments [1]. Civil engineers might study a sediment's texture to find out density, porosity and permeability [5] which will translate to strength under load. The study of sedimentological data is also fundamental in fields like Petroleum Engineering, Geochemistry, Hydrology [17] and a number of others.

Furthermore, there’s a current need for sediment sample data in Education. Access to sedimentological data can greatly improve numerous sedimentology courses by creating more studying materials [62]. Before university it is also important to get students interested in the field in more general geology courses. Another important aspect is the support of interested members of the public which currently might have non-trivial amateur collections [52, 53, 32]. There’s a clear need for a system which allows access to this kind of data.

1.1 Problem Statement

Having established the need for open access to sedimentological data, some problems which stem from the fact that there’s currently no easy to use system can be made explicit.

A recurring problem is the lack of data from field work already performed. Several researches will perform duplicated field and laboratory work in samples from the same location because there’s no way to know samples were already collected and that the data exists elsewhere. For the data that exists, a non-trivial amount is stored in old fashioned paper format or local spreadsheets in a department computer [18], [25]. There’s a very high cost involved in reproducing results since it is difficult to validate where samples were collected, where they are stored and exactly what tests have already been conducted. There’s no way to create “big picture” research (either in terms of large timespans, large areas or both) unless a given research group incurs in a single concerted effort. Even in this case, there’s no way for outside groups to participate and contribute to the effort.

In educational environments, such as universities, professors struggle to obtain enough quality sam-
ples to ensure students can have “hands-on” experience with the methods they need to learn. No online tools exist for students to study sediment samples at home, all of the work has to be done on premises and dissemination of materials happens by sharing paper copies of sediment data.

Even in secondary education access to samples, data and information is needed. It is expected that Portuguese teachers of the Biology/Geology course take their students to perform field work and collect sediment samples, analyze them in a laboratory setting and make conclusions [38]. This is very hard, as demonstrated by inquiries to the teachers themselves [4]. Difficult to organize field trips to perform field work, lack of dedicated equipment to perform analyses are barriers even to informed teachers with Geology backgrounds. There’s a harder problem though, more than half of the teachers have no formal training in Geology [4, p.6], and give lectures based on knowledge picked up along the way since they graduated in Biology related fields. In all of these cases an open platform where sample data is available would allow these teachers to give their students some exposure to the materials they are required to teach. Furthermore, the existence of this open data would foster the creation of more public educational material.

More specific to the technology needed to develop such systems, there’s a steep learning curve to penetrate the geospatial field. There’s an abundance of standards, protocols and proprietary tooling which one needs to learn and grow accustomed to. Many of these proprietary tools are very expensive to acquire and maintain, which make the usefulness of an open system making use of free open source tools much more evident.

1.2 Objectives

With the purpose of solving the presented problems, a system in which sedimentological data can be stored, shared, analyzed visualized and reused will be created.

A set of general goals can be described as subjective objectives:

1. Having the knowledge that research exists – facilitating the discoverability of data.

2. Preventing the duplication of efforts regarding sample collection in the field.

3. Creation of global communities with shared research interests and facilitated collaboration opportunities.

4. Facilitating the reproducibility of results by other researchers by:

   (a) Returning to the geographical location, collecting similar samples and performing the needed tests.

   (b) Arranging the physical sharing of samples when geological changes on the field or lack of financial funds make re-collecting samples undesirable.

5. Allowing new research from combined data samples and patterns in large data volumes.

6. Allowing new research from the availability of historical data samples.
7. Allowing the education community, policy makers, the private sector and society in general to study, make decisions or take action based on the available information.

We aim to build a sediment database, publicly accessible through an online web application. Users will be able to add information to this public database. Universities with research samples, museums or enthusiasts with private collections will be able to store their data in this system in a more organized and easy to access way. Data will be openly accessible according to user defined permission models and users will also be able to search and visualize theirs and other users’ sets of data. To accomplish these goals a set of technical requirements were identified and are presented as fundamental to reach them. These requirements are what we propose to create with this thesis:

1. Design of a data model capable of representing a typical sedimentological sample, analyses and tests.
2. Implementation of the data model in a data store capable of manipulating geographic information alongside typical data types.
3. Design and implementation of a permission model allowing users to differentiate between private, restricted, and public data, in order to facilitate collaboration mechanisms in all stages of research.
4. Design and implementation of an API to access the data.
5. Implementation of importing and exporting tools to ease the integration of users’ existing work flows into the web application.
6. Design of a User Interface for users to consume the stored data.
   (a) Design of a web application architecture capable of serving the data to users (webserver, database server, search server...).
   (b) Development of deployment tools to create the designed architecture (host setups, program updates...).
   (c) Development of the web application.

Furthermore, this system should be extensible and customizable. We realize that the amount of possible sediment analyses and tests are quite large and will no doubt grow and adapt as time goes on. Communication protocols might also be upgraded to deal with future platforms and allow new integrations.

With this in mind we aim to build a open platform that can be extended as time goes on, serving as a solid base for further developments.

1.3 Document Outline

The rest of this document will present the process that a sediment sample goes through and the current tools geologists use to help with it (Chapter 2). Chapter 3) presents the technology stack that is
needed to create an online accessible database, a website and the advantages of each tool analyzed to accomplish what we propose.

Afterwards we present our actual solution describing the software used, the architecture and data model designed and the actual development (Chapter 4). Finally (in Chapter 5) we summarize the work performed and conclude with possible future work.
2. State of the Art

In this chapter we will present the ways in which samples are collected and processed. We will show some of the more common analysis made on the samples and what needs to be recorded upon archival. This study is needed to understand the requirements and the final operations of the SandDB platform.

Next we present existing protocols, standards and libraries to deal with geospatial data in order to give a clear picture of what is possible with existing technology and how common functionalities are currently implemented. This section is focused on open-source solutions.

2.1 Sample collection, archival and analysis

It is important to understand the whole process of our interaction with sediments in order to develop a platform which can assist us in a useful way. Sediments are usually collected in the field in order to bring them to a laboratory where further tests can be performed on the samples. As such, there are protocols that have to be followed and the methods of collection have to be carefully reported. Sometimes though, there are inconsistencies in the data. Existing samples might have been collected at an earlier date and no records can be found. They might also be collected by laymen enthusiasts and so this data may be absent. Afterwards, in the laboratory there are a number of tests that may be performed, or none at all. Tests might have different outputs based on the tools and methods used or its intended purpose. Finally, it is important to differentiate between a test run by a highly trained and expensively equipped research team and a weekend collector. It’s easy to imagine that data is usually not consistent. This will be a recurring theme in this work: data is usually not uniform nor organized, and we have to try and make the most of it.

Nonetheless, geologists already collect a lot of data and usually have standardized procedures to follow regarding what data should be kept. In the following sections we will describe what these data points are without being exhaustive, covering what we believe to be the most common subset of techniques and procedures, mostly from [25].

2.1.1 Collection and Archival

Table 2.1 presents the information important to make note during collection activities. Additional information such as a description of the sample, a preliminary classification or the purpose for which the sample was collected might also be considered. Similarly, upon archival of a sample it is important to make note of the following presented in table 2.2.

2.1.2 Analyses

While not trying to exhaustively present every possible laboratory test done to sediment samples, we will present what we believe is a representative subset, in order to paint a clear picture of the data which
needs to be stored in a software system.

### Sieving

Sieving is used to separate sediments by size portions to estimate the grain size distribution as well as a preparation for other types of analyses. The data gathered from sieving is presented in table 2.3.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Initial weight</td>
<td>Float</td>
</tr>
<tr>
<td>Final weight</td>
<td>Float</td>
</tr>
<tr>
<td>Grain weights by sieve</td>
<td>Pairs of Floats</td>
</tr>
</tbody>
</table>

Table 2.3: Overview of sieving data.

### Heavy mineral separation

Heavy mineral separation is a method used to identify the relative percentages of heavy minerals (density greater than $2.9 g/cm^3$). It allows us to determine the origin and history of sedimentary rocks. It is usually performed with a dense (heavy) liquid in a centrifuge or separatory funnel. Magnetic minerals can then be separated with a hand magnet. The information resulting from this test is presented in table 2.4.

### Munsell color

Munsell color is a standard way to describe soil color using the Munsell Soil Color Chart. It is used to better classify sediments and sometimes get a clue about their chemical compositions. See table 2.5.
<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Initial weight</td>
<td>Float</td>
</tr>
<tr>
<td>Final weight</td>
<td>Float</td>
</tr>
<tr>
<td>Light minerals weight</td>
<td>Float</td>
</tr>
<tr>
<td>Heavy minerals weight</td>
<td>Float</td>
</tr>
<tr>
<td>Magnetic minerals weight</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.4: Heavy mineral separation data.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Hue</td>
<td>Text</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
</tr>
<tr>
<td>Chroma</td>
<td>Float</td>
</tr>
<tr>
<td>Moist or Dry</td>
<td>Text</td>
</tr>
</tbody>
</table>

Table 2.5: Munsell color data.

Moisture content

Moisture is the ratio of the water weight in a sample to the weight of the dry sample. It is important to know groundwater recharge and soil chemistry. See table 2.6.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Water weight</td>
<td>Float</td>
</tr>
<tr>
<td>Wet soil weight</td>
<td>Float</td>
</tr>
<tr>
<td>Dry soil weight</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.6: Moisture content data.

Atterberg limits

Atterberg limits are used to classify the fine-grained portion of a soil. These limits are defined as the plastic limit and the liquid limit and they are used to load bearing, compressibility, shear strength, swelling potential and sensitivity to change. A device called Casagrande cup is used to measure a number of blows until two sides of a soil touch each other, hence this is an important data point to take note. See table 2.7.

Soluble salts

By measuring the conductivity (S/m) of a water extract obtained from a pore water extraction method we can estimate the amount of soluble salts in a soil. See table 2.8.
<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Number of blows</td>
<td>Int</td>
</tr>
<tr>
<td>Moisture content</td>
<td>Float</td>
</tr>
<tr>
<td>Liquid limit</td>
<td>Float</td>
</tr>
<tr>
<td>Plastic limit</td>
<td>Float</td>
</tr>
<tr>
<td>Plasticity index</td>
<td>Float</td>
</tr>
<tr>
<td>Liquidity index</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.7: Atterberg limits data.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Conductivity</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.8: Soluble salts/conductivity data.

**pH**

Soil pH is a characterization of the hydrogen ion concentration. It measures the alkalinity or acidity of a sediment. It can affect nutrient availability, water content and toxicity of soils. See table 2.9.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>pH</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.9: pH data.

**Total and organic carbon**

Determination of carbon contents in a sample can be made by combustion, or the loss-on-ignition test. Soil carbon improves its water holding capacity, fertility, nutrient retention and the capacity to protect groundwater form cation contamination. See table 2.10.

**XRD Analyses and X-Radiography**

These are non-destructive techniques for qualitative and quantitative determinations of materials and internal structures of minerals and sediments. They provide either X-ray diffraction spectra or full scale sample radiographs. These can be stored in a PDF or other file for further consultation.

### 2.2 Geospatial protocols, standards and libraries

An overview of existing protocols, tools and software used to deal with geospatial data will now be presented. There will be a large number of tools that won’t be presented but an effort is done to give a general picture of what is currently available and widely used. First we’ll present existing protocols, then software built upon these specifications and lastly we’ll present some full fledged systems which make use of all the existing small parts.
<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample identifier</td>
<td>Text</td>
</tr>
<tr>
<td>Total carbon</td>
<td>Float</td>
</tr>
<tr>
<td>Inorganic carbon</td>
<td>Float</td>
</tr>
<tr>
<td>Organic carbon</td>
<td>Float</td>
</tr>
<tr>
<td>Loss on ignition</td>
<td>Float</td>
</tr>
</tbody>
</table>

Table 2.10: Total and organic carbon data.

2.2.1 Open Geospatial Consortium

The Open Geospatial Consortium is a standards organization which develops, implements and maintains the specifications and standards for open geospatial tools. Many of the tools presented henceforth have originated from OGC with the notable exceptions of GeoJSON and TopoJSON.

2.2.2 Geospatial data representation formats

**SRID**

Different representations of the Earth can be made by using different models for the sphere. All these models will have some degree of error, representing coordinates, but all aim to be as precise as possible. To maximize accuracy when using coordinates, one can opt to represent only a subset of the Earth and optimize a model for that specific region, obtaining much better results. Along the years, many entities have developed thousands of spatial coordinate systems. In order to unambiguously identify these
systems, a Spatial Reference System Identifier (SRID) is used. SRID's are usually stored in databases with geospatial features by their datum, coordinate system and map projection.

Geography Markup Language

Geography Markup Language (GML) [11] is an encoding specification for geodata in XML. With this specification appeared a way to embed both properties and the geometry of geographic features in XML documents, allowing for the adequate transport and storage of geographic information in XML. Examples of such properties are features, geometries, coordinate reference systems, topologies, units of measures and others.

```xml
<geometry>
    <gml:Point srsName="EPSG:4326">
        <gml:coordinates>-7.63086,37.31902</gml:coordinates>
    </gml:Point>
</geometry>
```

Listing 2.1: Example of geometry GML representation in a XML document.

GeoJSON

GeoJSON [28] is an open standard extending JSON and defining several types of JSON objects to represent data about geographic features, their properties, and their spatial extents. It supports the Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon types. Lists of geometries are represented by a GeometryCollection. Geometries with additional properties are Feature objects. And lists of features are represented by a FeatureCollection.

```json
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [2.0, 3.0]},
        "properties": {"color": "blue"}
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString", "coordinates": [[5.0, 5.0], [10.0, 10.0]]
        },
        "properties": { "color": "red", "thickness": 2.0 }
    }]
}
```

Listing 2.2: Example of geometry GeoJSON representation in a JSON document.
TopoJSON

TopoJSON [8] extends GeoJSON and eliminates redundancy in the description of geometries. For this much more efficient representation, line segments - or arcs - are defined and then referenced by possibly multiple different geometries. Another difference is that Features and Feature collections are collapsed into their geometry types and the Feature properties are stored directly in the geometry object. This way both identifiers and properties are maintained but stored in a more compact way. Another possible optimization which is made possible by the topological representation is that we can simplify geometries’ edges and significantly reduce file size at the expense of map edge precision.

```json
{
    "type": "Topology",
    "transform": {
        "scale": [0.036003, 0.017361],
        "translate": [-180, -89.998925]
    },
    "objects": {
        "aruba": {
            "type": "Polygon",
            "arcs": [[0]],
            "id": 533
        }
    },
    "arcs": [
        [3058, 5901], [0, -2], [-2, 1], [-1, 3], [-2, 3]
    ]
}
```

Listing 2.3: Example of geometry TopoJSON representation in a JSON document.

2.2.3 Geospatial Libraries

A number of different software libraries exist to deal with geospatial data, provide efficient ways to perform computations, calculate intersections, distances and a range of other operations. Furthermore, tools are needed to perform operations in different world projections, convert between existing data formats while maintaining correctness of the data in question. In this section we present a set of the most used open source libraries which regularly appear and are used by most Geographic Information Systems (GIS).

Geometry Engine Open Source - GEOS

GEOS [45] provides a geometry model exposing several geometry classes, geometric functions, spatial structures and algorithms as well as I/O capabilities for compatibility with data formats such as GML. GEOS is included and used by a large number of geospatial tools and systems ranging from PostGIS to
Google Earth.

**Geospatial Data Abstraction Library - GDAL/OGR**

GDAL [44] is a translator library for raster and vector geospatial data formats by the Open Source Geospatial Foundation which embeds GEOS. It uses a raster and vector abstract data models and is compatible with a large number of data formats. It also provides tools for data processing, translating between coordinate systems and a range of several other command line utilities.

**PROJ.4**

PROJ.4 [48] is a library for performing conversions between cartographic projections. Generically, to represent locations on a surface in a plane a projection is needed. These projections will all introduce different distortions in the represented features and as such, different projections are able to retain different properties better than others. With PROJ.4 converting between different projections becomes a non-issue and as such this library is used by many other tools.

### 2.2.4 Geospatial data storage

Geospatial data storages or geodatabases are special databases optimized to deal with data representing features in a geometric space. These optimizations usually refer to the querying of data, by allowing complex queries based on distances, intersections, overlaps and others. In order to improve the efficiency of these kinds of queries, data can also be stored in specialized structures. Since these special modifications to data storage and querying can be added on to regular databases, they are usually added as special plugins or modifications to existing well-known systems. We will present some of them, as well as their advantages and disadvantages.

**SQLite + SpatiaLite**

SQLite [27] is an embedded database, writing directly to a file which can simply be accessed by an application. By not having a client/server architecture it is much simpler to use, requiring much less configuration and having much less latency for not having to communicate over a socket. On the other hand SQLite doesn’t have a separate daemon to handle locks and leaves this to the operating system at the file level. As such SQLite isn’t appropriate for multi-user systems where multiple simultaneous writes would incur in global locks and slow down access to data. While SQLite supports some spatial capabilities, SpatiaLite [23] expands these by providing a spatial engine capable of advanced spatial queries and multiple map projections.

**PostgreSQL + PostGIS**

PostgreSQL [56] is a very stable RDBMS using a client/server architecture. It supports a number of features and has a very large open source community of developers and users. It's one of the most popular open source database systems. The PostGIS extension adds support for geographic objects
allowing spatial queries to be run in the database. It includes types for geometry, geography, raster and others. It's compliant with OGC and a part of OSGeo foundation. It is the de facto choice for spatial applications for its feature completeness, stability and support of other mapping standards such as WMS, WFS, WFS-T, WCS, WPS and WMTS. It also can be used with GDAL and others to easily import and export data.

MySQL

MySQL [43] is an RDBMS which also uses a client/server architecture like PostgreSQL. Despite being one of the most popular database systems, MySQL has historically been lacking in the spatial department. Support for complex spatial queries usually lagged PostGIS implementations and generally behaved much less reliably. On top of these problems, MySQL's spatial capabilities are not OGC compliant, only support bounding box operations (can’t filter by different geometries and polygons, just rectangles) and so a lot of spatial functions will return wrong results if one is not expecting that every geometry will be bounded by a rectangle. In addition, there is no default support for spatial indexes (R-trees), which makes for a performance hit when comparing to PostgreSQL.

MongoDB

MongoDB [39] is a document database. It has some advantages regarding scaling with options for automatic sharding of data over several servers, and out of the box includes functionality for replication and high availability. By using documents instead of regular tables and rows, MongoDB avoids the use of expensive joins when querying. Furthermore, there is no concept of data schema which means there are no migrations when the non-existing schema changes, one can just change the type of stored documents on the fly. Regarding spatial data, MongoDB only supports surface types (2D), either spherical or plane. When using sphere types it only supports the WGS84 datum. Data is stored as GeoJSON objects with its default features. Querying is also very limited with only three available operators, Inclusion, Intersection and Proximity. As such MongoDB might be a good choice for very simple applications, and any sophisticated computations will have to occur on the application side.

Evaluation

Summarizing all the points above from our analysis of different database systems with a focus on geospatial features we present a comparison of their capabilities in order to more easily visualize how they match up. See table 2.11.

<table>
<thead>
<tr>
<th></th>
<th>Multiple User</th>
<th>Relational</th>
<th>Spatial Queries</th>
<th>Map Projections</th>
<th>OGC Compliant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatialite</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PostGIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>✓</td>
<td>✓ (limited)</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>MongoDB</td>
<td>✓</td>
<td>✗</td>
<td>✓ (very limited)</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 2.11: Data store feature comparisons.
2.2.5 Protocols and methods for geospatial data processing and presentation

Web Map Service

The Web Map Service (WMS) [15] allows a client to display georeferenced map images from a server using data from a GIS database. These map images can, in addition, contain an overlay of vector graphics in the form of geometric shapes and even text. These overlays, or legend, give a visual guide to map elements. Another way of thinking about the raster images returned by WMS is of overlays themselves on top of a previous map image, for example overlaying a satellite image of a certain area over a regular map (Figure 2.8).

![WMS Overlay Example](image)

Figure 2.2: A raster image requested with WMS overlayed on an OpenStreet Map.

Web Coverage Service

The Web Coverage Service (WCS) [13] provides an interface to access multi-dimensional coverage data. Coverage data is an extended feature which has different (and possibly multiple) values at each location. This allows for the representation of properties that vary along dimensions. Examples of these kind of properties are elevation, pollution, pressure and others. While WMS can return images, WCS can be used for complex modeling by representing variable features along the different dimensions. WCS supports some extensions which also allow for more complex querying and processing.

Web Map Tile Service

Building on the WMS specification, the Web Map Tile Service (WMTS) [16] was created to address problems in rendering WMS maps and serve them in an acceptable amount of time. By creating a service where previously rendered tiles can be served, performance is greatly enhanced and the system becomes much more scalable. Most consumer online mapping systems currently use some variant of this standard, by serving responses in tiles.
Web Feature Service

The Web Feature Service (WFS) [14] proposes an interface for describing data manipulation operations in a distributed manner on geographic features by using HTTP. A feature is described by a set of properties where each property can be thought of as a name, type, value tuple. Geographic features are those that may have at least one property that is geometry-valued. Basic WFS supports both simple and complex queries based on spatial and non-spatial constraints in the data, allowing for querying and retrieving features.

An optional extension to WFS, WFS-T, or transactional WFS creates an interface which supports all CRUD (Create, Read, Update, Delete) operations on feature instances.
2.2.6 Web Mapping

In order to present and share maps and spatial data in the web, a lot of engineering power had to be devoted to building interfaces and being able to properly load geospatial data into a user’s browser. With the appearance of Google Maps and their API [26], it suddenly became much easier to integrate a map into a web page. By using their map tiles and processing infrastructure one can (with some limits) easily use maps on the web. After the appearance of Google Maps some open source projects have appeared to also allow web mappings. We present the two most used solutions.

OpenLayers

OpenLayers [47] was the first open source widely used library to display maps on a web browser. The project is a member of OSGeo and has a very large number of features. It is compatible with OGC formats presented in 2.2.5 and a number of others.

Leaflet

Leaflet [2] appeared five years after OpenLayers with the goal of making a web mapping library with a more easy to use API. Along with that goal, Leaflet has a much smaller footprint and supports many of the OpenLayers project functionalities via a plugin system. Because of its much smaller footprint it is also a much better choice for mobile compatible applications.

Evaluation

Table 2.12 compares the presented web mapping alternatives.
<table>
<thead>
<tr>
<th></th>
<th>Open-source</th>
<th>Extensible</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Maps</td>
<td>✗</td>
<td>✗</td>
<td>~500kB</td>
</tr>
<tr>
<td>OpenLayers</td>
<td>✓</td>
<td>✗</td>
<td>~700kB</td>
</tr>
<tr>
<td>Leaflet</td>
<td>✓</td>
<td>✓</td>
<td>~65kB</td>
</tr>
</tbody>
</table>

Table 2.12: Mapping solutions for web clients.

### 2.2.7 GIS Desktop and Server Applications

#### QGIS

QGIS [49] is an open source GIS desktop application for creating, editing, visualizing, analyzing and publishing geospatial data. It is a member of OSGeo and compatible with most operating systems (Mac, Linux, BSD and Windows). It also features a server where you can configure how to publish your data as OGC compatible WMS and WFS services. It also includes a web client to consume services exported on the web. It allows integration with other open-source GIS software such as PostGIS and MapServer and includes a plugin framework to extend its capabilities. It is a full featured geographical information system competing with the proprietary and closed source ArcGIS.

![QGIS desktop](image1)

![QGIS server](image2)

![QGIS web client](image3)

Figure 2.6: QGIS desktop application.

Similar to the QGIS server already referred in 2.2.7 other servers exist to present geospatial data through the internet. The server usually has the ability to connect to a number of data sources, indexes the data in some way that allows it to be queried and exported in a number of standardized formats. We will present three different existing open source solutions.

#### MapServer

MapServer [41] was the first open source GIS server. Originally developed with funding and for NASA, it has been releasing new versions since 1997. It is a member of the OSGeo foundation and compatible with OGC standards. It integrates with a number of data sources including PostGIS, supports reprojections and leverages GDAL and PROJ.4. It can present vector and raster data, and has support for complex spatial and attribute based querying. Additionally it includes MapCache which is compliant with WMS and WMTS and serves as a tile caching server. It's interface is minimalistic which makes for a steep learning curve.
GeoServer

GeoServer [46] offers many of the same functionalities of MapServer. It is more recent than MapServer but its major differentiation is the front end administration panel it offers, which makes it much easier to start using and allows for a better experience exporting data. Additionally GeoServer has support for transactional WFS (WFS-T) and WPS which make it possible to users to alter the database through its API. Its implementations of WFS and WCS are the reference implementations of these standards by OGC.

![Figure 2.7: GeoServer import and export formats and protocols.](image)

Evaluation

Table 2.13 compares the presented GIS server applications.

<table>
<thead>
<tr>
<th></th>
<th>Project Start</th>
<th>WFS/WFS-T</th>
<th>Querying</th>
<th>Administration</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapServer</td>
<td>1997</td>
<td>✓/X</td>
<td>SQL</td>
<td>X</td>
</tr>
<tr>
<td>GeoServer</td>
<td>2003</td>
<td>✓/✓</td>
<td>OGC Filters</td>
<td>Web GUI</td>
</tr>
</tbody>
</table>

Table 2.13: Comparison of MapServer and GeoServer features.

2.2.8 Overview of geospatial software by features

In table 2.14 we compare all the presented software tools by features. It shows where each piece of software fits the overall picture of creating a geospatial enabled system. It is sometimes hard to step back and understand all these pieces separately because many of them embed others in order to achieve the same results and so in this table we try to summarize all these server-side solutions.
<table>
<thead>
<tr>
<th>R/W (WFS-T/WPS)</th>
<th>Read-Only (WFS/WMS/GML)</th>
<th>Map Creation</th>
<th>Reprojection</th>
<th>Conversion Formats</th>
<th>Geometry support</th>
<th>Spatial Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QGIS Map Server</td>
<td>PROJ.4</td>
<td>GEOS</td>
<td>GDAL/OGR</td>
<td>PostGIS</td>
<td>PostgreSQL</td>
</tr>
<tr>
<td></td>
<td>GeoServer</td>
<td></td>
<td></td>
<td></td>
<td>Spatialite</td>
<td>SQLite</td>
</tr>
<tr>
<td></td>
<td>MapServer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MongoDB</td>
</tr>
</tbody>
</table>

Table 2.14: Overview of server-side geospatial open source software by features.
2.3 Web applications using GIS servers

Since 1996 when NASA funded the development of MapServer with the purpose of making its satellite imagery available to the public, the proliferation of geospatial web applications has increased dramatically. Currently there's a large number of implementations using some set of the standards, protocols, libraries and tools we've presented so far. In this section we present some prominent examples of such implementations.

2.3.1 Governmental web applications

Government agencies such as the United States Geological Survey (USGS), the British Geological Survey (BGS) or the Portuguese Geographic Institute (IGP) have developed programs to share geographic data and metadata with the public.

![Image](image1.png)  
(a) USGS Sediment Data Portal [58]  
(b) BGS OpenGeoscience [9]  
(c) LNEG’s geoPortal [33]

Figure 2.8: Online government programs for geoscience.

There are already numerous types of data one can access through web services and even specially developed mobile applications [3]. By allowing users of their websites to search, visualize and download data these platforms are hugely beneficial. Lots of effort has been dedicated in creating systems which can harbor totally generic metadata on top of geographic information (points, areas, borders...). One such system which is used by entities such as the United Nations, European Space Agency, and numerous others [57] to gather information and make informed decisions is the GeoNetwork Opensource [42]. This system uses GeoServer as an underlying server and provides extra interfaces and administration pages like a more conventional Content Management System (CMS) in order to allow users to easily create a catalog of geospatial data. In these systems there’s topological information, fauna and flora, hydrological information, forest fires and others. Recently these systems were critical in the fight on the 2014 West Africa Ebola outbreak. They were used to map medical locations, affected areas, clean water sources and many other important geospatial features.
Some of these institutions have also made these repositories open for educational purposes [59]. There’s even examples of laboratory activities developed and proposed by the USGS to be used in a classroom environment [60]. These have some requirements on either being able to collect new samples or already having them in store to use during the activity. An online repository with open sample data would be a great supplement to these activities, either in a student’s autonomous study as well as in making sure classrooms have access to sample data when they can’t obtain physical samples.

By being so generic and allowing data from all kinds of sources, these systems are usually limited to the discovery of information. To process this data one usually has to download it and presumably import it into another tool. There’s also the problem that while trying to serve everyone, we can’t usually serve everyone well.

2.3.2 Geological systems focused on sediment storage and analysis

A more specialized online platform was created to deal with specific sediment samples, SedDB [34]. It currently hosts a database with over one hundred thousand samples and associated metadata. It is developed by the Integrated Earth Data Applications (IEDA) Research Group and funded by the US National Science Foundation (NSF). Its goals are much in line with those of this thesis. It aims to have an accessible database of marine and continental sediment samples to help research and education.

This platform allows for some of the functionalities we propose. It has a publicly available database, with search, visualization and there’s the ability to contribute data. To contribute data one has to get an International Geo Sample Number (IGSN) [31], which is a 9-digit alphanumeric code where the 3 first digits are a unique user identifier and the rest are random. The user has to register with GeoPass which is an external service that provides a profile that can be used to submit data in several IEDA platforms. The user then goes to the System for Earth Sample Registration (SESAR) [29] and can upload a sample. Users are encouraged to treat sample uploads as journal submissions, by making sure they are in their final forms [30]. Search can be done either in the SESAR platform, by IGSN or on SedDB but there’s not much data associated with the samples, often times just providing the location, method of collection and chemical composition of the sediments. The samples are also very much tied to specific published articles and no other associations or further test results can be added after the first submission.

As we can see in search results of these platforms, there’s usually no further work being done by the
platform to make visualizations data more useful to the user. A map being displayed with the sample
collection location coordinates is the most elaborate visualization we get.

By defining the tool as a sediment database we greatly reduce the number of possible laboratory
tests and resulting data types generated. Even so, the tools generally don’t display things like data plots
or even allow for the uploading of accompanying PDF files in order to share generic test results. Sharing
photos of the samples can also help in making a mental model of the sample, by learning to visually
identify different types of sediments by color, size, texture and other features. It even allows for research
like trying to quantify these features automatically with computer vision algorithms [35] which could then
be rated for accuracy by using the accompanying data.

2.3.3 Other local tools

Further data analysis is currently performed locally by the researcher in either self developed tools or
the more prevalent spreadsheets. A common type of analysis in sediment research is sieving to get
a statistic distribution of grain sizes in a given sample. From this type of analysis, much information
[36, 24, 22, 51] can be extracted from both the sample and its environment and transportation. To get
this information, geologists plot the distribution in a number of slightly different ways and run a number
of equations through the data. Since this work can be quite laborious, the currently accepted method is using a hand crafted base spreadsheet (GRADISTAT) [7] where you input your sample data and check the outputted graphs and results. This is a clear example of results that could be generated by an online tool that already has the data. A number of other common tests to samples follow this same principle.

![Sample Statistics Table](image)

**Figure 2.12:** The GRADISTAT's spreadsheet data input.

Ultimately, one of the more glaring faults found was that none of the platforms we could find had any concept of community. There's no interaction between consumers and producers of data, and no discussion can be had through the platform. Even in the best case of SedDB, the interface is quite uninviting to an unexperienced user, and there's no way for users to continually improve the database through discussion and collaboration. We believe that by allowing test results, photos and discussion, it will be easier to collaborate on data categorization and to provide and receive feedback.
Figure 2.13: GRADISTAT’s generated gravel sand mud diagram.
3. Supporting technologies and tools

Previously in this document’s objectives we specified the requirements for this work. Afterwards in section 2.2 we investigated, described and compared existing standards, protocols, libraries and tools that support geospatial features in a software product. In this following section we will analyze requirements of a generic web application and compare them among themselves while considering the final goal of integrating with geospatial tools. These will finish outlining the possible software choices on which we will build upon and pave the way for a presentation of the decisions that were made requiring the project’s global architecture.

3.1 Architecture requirements for a Geospatial Web Application

Any generic web service needs some pieces of software from which we cannot deviate. End users will connect to the service typically using a browser or through an API which is served by a web server. In order to have a dynamic application and serve more than static files the web server will have to connect to a programming language through an standard interface, or in some cases this kind of functionality might even be embedded in the server (for example Lua embedded in the nginx web server), or be the server itself (for example node.js). Requests to a programming language are handled and a result is returned. In order to do this a set of libraries can be put together as a web framework to speed up development. These can facilitate access to databases, handle request routing and deal with other common tasks. For some kind of persistence a database of any sort can be used. If we aim to provide search abilities we might plug in a search server which indexes database data and prepares it for search. For long running processes that can be made asynchronously we might plug in a distributed task queue system, and so on. In the case of geospatial systems, another system is usually introduced. A special kind of webserver that understands OGC standard queries and replies appropriately is exposed both to the application and the end user. This puts the geospatial heavy lifting related work outside the main application which only concerns itself with geo-agnostic requests such as authenticating users and creating the context for the interactions with the geospatial data.

3.2 Front-end requirements for a Geospatial Web Application

A web application front-end consists on what the user interacts with in a browser. Generally the application will serve HTML formed by some kind of templating engine and from then on Javascript will allow for any dynamic presentation of content. These interactions can be made easy by the use of Javascript libraries to abstract browser and operating system dependent inconsistencies and to ease some common tasks. Additionally, in the case of a geospatial web application we will need some way to receive and present geospatial data to the user. This will include a mapping display with the ability to present features as layers, cluster points, annotate geometries and other common user interaction features on a...
mapping display. Another front-end use of the service is through an API in which case the response will usually just contain data in some common format like XML or JSON.

Figure 3.2: Generic geospatial enabled web application front-end.
4. **SandDB**

To approach the problems we presented so far we decided on the implementation of a web application from here on referred to as SandDB. In this chapter we will present the decisions that were made regarding underlying technology and the features that were implemented.

Research was done on the relevant data types of a collected sample before any processing. We also identified a set of tests which we presume to include the majority of use cases. The decision on the kind of data store used was centered on the assumptions that the application would be a read-heavy, multi-user system.

4.1 **Supporting technology**

**Database**

A database management system (DBMS) had to be chosen from existing open-source solutions. Furthermore, by acknowledging that most of the data would be associated with geographic features, the chosen database would have to support a geographic information system (GIS) to facilitate the storage and querying of this kind of data. From our analysis of a subset of existing systems in 2.2.4, we can already exclude SQLite from the requirement of a multi-user system. Regarding multi-user environments with highly concurrent queries a NoSQL system like MongoDB would be the front runner, but since geospatial functionalities are very limited we also discard this system. Going on number of geospatial features and remembering that MySQL returns spatial queries only for rectangle bounding boxes and is not OGC compliant, PostgreSQL is the chosen system. This could already be predicted from our comparison in table 2.11 of needed features in data storage systems.

PostGIS [50] has numerous features, a very large community compared to other open systems, is more stable and has much more comprehensive documentation than any other we found. It is also the database back-end for a large number of software products, from ArcGIS to the OpenStreetMap website, to the Uber mobile and web applications to name just a few.

**Web Framework**

For users to access data, an API has to be designed on top of our data store. Early on we decided on a RESTful (Representational state transfer) architecture [37] [21], and needed to find a framework with the ability to interact with a generic database driver, which allowed for authentication and authorization methods, had good documentation and had already reached a stable release. Additionally it would be helpful to use a stable and widespread underlying programming language, both to guarantee good library support and to better our chances of getting future project maintainers. Since we needed to develop this service with direct feedback from our end users we also needed to decide on a web framework for the primary website. Stemming from that fact that it had extensive library support to deal with spatial data,
plotting and statistic analysis, which would greatly speed up development, Python was selected as the primary programming language for the project. In table 4.1 we compare the two most popular Python web frameworks.

From there, we chose the Django Web Framework [19] since it had the largest feature set, most complete documentation and an active community and development team. Because it fulfilled all of our previously stated requirements, we chose the Django REST Framework [10] to implement our REST API. Most significantly it provided a web browsable API which greatly facilitates the development process.

<table>
<thead>
<tr>
<th>Object Relational Mapping</th>
<th>Community Size</th>
<th>Geospatial Features</th>
<th>Philosophy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Django ✓</td>
<td>+++</td>
<td>GeoDjango included</td>
<td>Batteries included</td>
</tr>
<tr>
<td>Flask X</td>
<td>++</td>
<td></td>
<td>Highly modular</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of Python web frameworks

**Programming language Interface**

A web server is not enough to serve our Python web application and all needed static files (photos, css, javascript files...). We need to use an application container which implements Web Server Gateway Interface (WSGI) [63] to allow messages to be passed from a web server to our Python application. One
such container, uWSGI [61] is a fast, self-healing and developer/sysadmin-friendly application container server coded in pure C. Django and the REST Framework extension will be served from uWSGI.

**Web Server**

A web server is needed to expose the application to the Internet. Both the nginx [40] and Apache [54] web servers were analyzed. Both servers have similar capabilities, community and support. Trying to choose based on other capabilities, such as speed, is hard before having a developed application since benchmarks will be very dependent on the use case. Since this was the case we ended choosing nginx for its simpler configuration and its embedded uWSGI support. With nginx we can make communications between the web server and uWSGI go through a UNIX socket which uses uwsgi protocol which is more efficient than if not.

**Search engine**

The database is only accessed from the Django application. Django should interact with a full-text search server so we can provide users the ability to search the website in an efficient way. Two competing full-text search servers dominate in terms of features and community, Solr [55] and ElasticSearch [20]. Since no major limitations were encountered in any one of them ElasticSearch was chosen for its from the ground up distributed nature, and arguable ease of use and configuration.

**GIS Server**

Regarding the GIS Server, the needed functionalities were analyzed. At first the overwhelming amount of features they provided may entice us into using a full fledged solution and try and integrate as many of these features as we can. What was found out as the project went on though, was that only a minor subset of these features would be truly important for our specific use case. As such a decision was made to not use any external GIS server and implement the needed functionalities into the application. Since Django provides a modular way of developing "apps" with contained features if it ever becomes apparent that an external GIS server is actually needed, we shouldn’t incur in much added development effort to pull the functionality out.

### 4.2 Django architecture overview and request life-cycle

In order to explain the implementation of the data model and some of the decisions that were made, it is important to understand how all the moving parts of the Django web framework work together. A Django project starts by having a small amount of configuration files, a file with URL patterns and a small script to allow the framework to interact with a web server. There are applications, which are used to perform a particular task. As an example, if a website has a discussion board such as a forum, an application might be responsible for user private messaging, another for the forum threads, and another for registration. These applications can be re-used and shared, making them effectively like an extension, or plugin,
for Django. Inside each application folder are its models, views and templates, following the Model-View-Controller (MVC) architectural pattern. In Models the data is defined, in Controllers (Django views) actions are performed, and in Views (Django templates) we typically return HTML pages following some presentation logic.

```
|-- api
  |-- models.py
  |-- tests.py
  '-- views.py
|-- manage.py
'-- sandDB
  |-- settings.py
  |-- urls.py
  '-- wsgi.py
```

Listing 4.1: Django generic project structure with one application.

The way we define models is as Python classes, where each class maps to an SQL table and each class attribute to a table column. Querying happens through an ORM which is basically an abstracted way of interacting with different database backends with the same API. Controller logic in the views consists of either regular Python functions or Class-Based Views where code can be more easily shared by different pages making use of object oriented abilities. An overview of Django’s architecture is present in figure 4.3.
Figure 4.3: Django architecture overview. Adapted from mytardis.readthedocs.org/en/latest/architecture.html.

1. The URL dispatcher (urls.py) maps the requested URL to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all further steps, returning the cached version, instead. Note that this page-level caching is only one available caching option in Django. You can cache more granularly, as well.

2. The view function (usually in views.py) performs the requested action, which typically involves reading or writing to the database. It may include other tasks, as well.

3. The model (usually in models.py) defines the data in Python and interacts with it. Although typically contained in a relational database (MySQL, PostgreSQL, SQLite, etc.), other data storage mechanisms are possible as well (XML, text files, LDAP, etc.).

4. After performing any requested tasks, the view returns an HTTP response object (usually after passing the data through a template) to the web browser. Optionally, the view can save a version of the HTTP response object in the caching system for a specified length of time.

5. Templates typically return HTML pages. The Django template language offers HTML authors a simple-to-learn syntax while providing all the power needed for presentation logic.
4.3 Data modeling

4.3.1 Samples

We started to develop the data model by assuming a registered user would input samples and their properties into the system. The sample model is defined in a Python class in Django as shown in 4.2.

```python
class Sample(models.Model):
    user = models.ForeignKey(User)
    expedition = models.ForeignKey(Expedition)
    description = models.TextField()
    sandDB_code = models.CharField()
    retrieval_date = models.DateField()
    retrieval_location = models.CharField()
    latitude = models.FloatField()
    longitude = models.FloatField()
    retrieval_depth = models.IntegerField()
    retrieval_height = models.IntegerField()
    geopoint = models.PointField()
    objects = models.GeoManager()  # manager for geodjango
    institution = models.ForeignKey("Institution")
    collection = models.ForeignKey("Collection")
    creation_date = models.DateTimeField(auto_now_add=True)
    update_date = models.DateTimeField(auto_now=True)
```

Listing 4.2: The sample model.

A sample has an owner which is the original uploader of the sample. A user which has ownership of a sample (write privileges) is able to input further data and test results. There's also the concept of grouping samples. The first possible grouping is a Campaign which exists when a user collects one or more samples in an outing by performing some field work in a limited location and time-frame. Finally there are Classifications which are a kind of tagging system. A sample can have multiple classifications and samples can be searched, grouped and filtered by classifications. A site administrator can create any number of classifications types with possible values. These are implemented as ClassificationGroup's and Classification's in a way that's extensible by a non-developer user.

This sample model allows us to easily support more test data relating to the samples. We only have to identify common analysis and their output data, create a table which models it and link it to our Sample model by a foreign key. The first test we decided to support was the granulometry analysis since it is a widely performed operation with very useful output data. As we can see in figure 4.5, this test has a variable number of possible sieves, but the possible sieves are generally standardized. In this case we pre-fill the database with the default sieves and a user is then able to choose which sieves were used in a given test. Another simple analysis which was also implemented was the pH test, which again, consists simply in a table with a relation to the Sample and in this case a DecimalField representing the
actual pH value.

Users can also upload photos and generic PDF documents in order to support other tests which aren’t currently implemented in the data model. This demonstrates the ability to simply plugin data to the sample model and support more tests as they are deemed necessary. In the special case of photos there’s several types of photos which can be inserted by an admin and a user can then select which type it belongs to (for example x-ray vs microscope photos).

### 4.3.2 Users and permissions

An implementation of users and model level permissions is provided by the Django framework. We leverage these models and only needed to create a Profile model which will store extra information about a user. As for the model level permissions, they allow us to create groups of users and grant permissions based on group membership. These are useful to define users with administration or moderation rights.
Row level permissions can be implemented to allow several users to have different read/write permissions on samples instead of a basic single owner model. Row level permissions, or per object permissions, can be implemented in Django as an object aware authorization backend by linking users to generic "content types" (Django’s way to generically refer to data models). For this, an extension like django-guardian can be used, but a simple implementation could just consist on something like is shown in listing 4.3.

```python
class ObjectPermission(models.Model):
    user = models.ForeignKey(User)
    can_view = models.BooleanField()
    can_change = models.BooleanField()
    can_delete = models.BooleanField()
    content_type = models.ForeignKey(ContentType)
    object_id = models.PositiveIntegerField()
```

Listing 4.3: Simplistic per object permission model.
Figure 4.7: Django user model, sessions, authentication and permissions.
4.4  REST API design

Every endpoint returns data in the format specified by the HTTP request headers, which in the case of the browser is HTML. The format can also be specified by adding \( \textit{format = json} \) to the URL which will then return data in the JSON format. In order to exemplify our URL schemes we will use regular expressions (regexes) which are commonly used for pattern matching in strings and in this case are used by Django’s default URL routing code. The implemented REST API supports the GET, POST, PUT, DELETE, HEAD and OPTIONS HTTP verbs.

4.4.1  Read-only, or safe methods

Every resource answers to HEAD and OPTIONS. Every endpoint replies to a HEAD request exactly as it would if it had been a GET request but only with the response headers. OPTIONS requests are answered with the possible HTTP requests and what each of them expects from the user in terms of submitted data at that specific endpoint.

GET requests are used to retrieve single or lists of resources. The base URL endpoint for the sample objects in our API is matched by the example regex in listing 4.4 and returns a paginated list of samples.

```
^samples/$
```

Listing 4.4: Base URL to access samples in the database.

A specific sample can be reached by specifying its primary key as an identifier.

```
^samples/(\?P<\text{pk}>[0-9]+)/$
```

Listing 4.5: URL to access a specific sample.

<table>
<thead>
<tr>
<th>URL Style</th>
<th>HTTP Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^resource/$</code></td>
<td>GET</td>
<td>list</td>
</tr>
<tr>
<td><code>^resource/(\?P&lt;\text{pk}&gt;[0-9]+)/$</code></td>
<td>GET</td>
<td>retrieve</td>
</tr>
</tbody>
</table>

Table 4.2: Safe methods overview for a generic resource.

4.4.2  Methods with side effects

Contrasting with the previous request types, we also implement methods with side effects on the server. These are used to create, update and delete resources. We use POST to create resources by hitting the base resource URL. In order to update a given resource we PUT to its specific URL (using its identifier). Likewise, if we only mean to apply partial modifications to a resource we would hit its specific resource URL with a PATCH request. Finally we can delete a resource by making a DELETE request to its specific URL.

Together with the safe methods we get full CRUD (Create, Read, Update and Delete) functionality on our data models with these types of endpoints.
### 4.4.3 Overview of implemented endpoints

Using the methods presented above, we’ve implemented endpoints for the resources in our presented data model.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^users/$</code></td>
<td>Users of the system</td>
</tr>
<tr>
<td><code>^groups/$</code></td>
<td>Groups of users (used for model level permissions)</td>
</tr>
<tr>
<td><code>^samples/$</code></td>
<td>Samples inserted by users</td>
</tr>
<tr>
<td><code>^granulometries/$</code></td>
<td>Granulometry tests associated with samples</td>
</tr>
<tr>
<td><code>^campaigns/$</code></td>
<td>Field work grouping several samples</td>
</tr>
<tr>
<td><code>^photos/$</code></td>
<td>Photographies associated with a user or sample</td>
</tr>
<tr>
<td><code>^files/$</code></td>
<td>PDF files associated with samples</td>
</tr>
</tbody>
</table>

Table 4.4: Implemented API endpoints.

Adding endpoints to the API is very simple and consists in defining both a model Serializer and a ViewSet. Using the Sample class model presented in 4.2 we present both its Serializer and its Viewset. A ViewSet consists of a query to the database and a definition of its serializer class in case we want to customize the output JSON. The Sample serializer is one of the most complex in the project and as such is used to give a good example of available features. It defines a special hyperlink fields which link to the specific objects for the actual sample, its owner (User) and the expedition on which it was collected. It also embeds the Granulometry information inside. Additionally, we declare a \texttt{geofield} and inherit from GeoFeatureModelSerializer allowing us to turn the JSON object into a GeoJSON object instead, with a feature of type Point on the geopoint attribute and so allowing the Sample table to be consumed by any GIS service compatible with GeoJSON.

```python
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = serializers.UserSerializer
```

Listing 4.6: Sample ViewSet.
```python
class SampleSerializer(HyperlinkedModelSerializer, GeoFeatureModelSerializer):
    url = serializers.HyperlinkedIdentityField(
        view_name='api-sample-detail',
    )
    user = serializers.HyperlinkedRelatedField(
        queryset=User.objects.all(),
        view_name='api-user-detail',
    )
    campaign = serializers.HyperlinkedRelatedField(
        queryset=Campaign.objects.all(),
        view_name='api-campaign-detail',
    )
    granulometry = GranulometrySerializer(
        read_only=True,
    )

class Meta:
    model = Sample
    id_field = False
    geo_field = "geopoint"
```

Listing 4.7: Sample Serializer.

### 4.5 Django-wfs app

You might recall from section 4.1 that we made the decision of not using a full fledged external GIS server as a dependency because we believed it would add more complexity than needed for the features we had in mind. The features we did have in mind were interoperability with existing GIS services like the QGIS desktop application, where researchers usually load external web services. QGIS has support for GeoJSON data formats and so we could load this data in QGIS from our API as shown in figures 4.8.

(a) Add a vector layer in QGIS pointing at the API endpoint using GeoJSON.

(b) The loaded data samples using GeoJSON API.

Figure 4.8: How to use the GeoJSON API with QGIS.
Even though GeoJSON is quite good for modern web applications, it only serves very basic needs in QGIS. QGIS's support for GeoJSON doesn't include any kind of filtering or pagination of results which makes it quite unsuitable for any type serious usage. For these reasons, we decided to also provide users the ability to export Features (Samples) as a WFS 1.0.0 service. After searching for a suitable project compatible with Django or even Python without using a full external server, the decision was made to implement our own service. Our implementation of WFS was based on the official [12] Web Feature Service Implementation Specification, a 90+ page document describing all the capabilities of a compliant service.

Implementation started by creating the data models for the WFS application. The basis of this is the Service which exposes multiple FeatureType's. Each FeatureType can contain BoundingBox's and MetadataURL's. Generically each Service would represent a different kind of geospatial data, which in the case of SandDB, currently, only supports the Sample model. A FeatureType on the other hand can represent any grouping of objects under a Service. In SandDB for example this was used to autogenerate FeatureTypes for every User and every Campaign.

![Figure 4.9: Models for WFS implementation.](image)
As for the View organization of the Django-WFS app, we expose only one endpoint, like indicated by the specification, which is then queried using URL parameters. The underlying possible requests are described in table 4.5.

<table>
<thead>
<tr>
<th>Requests</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>getcapabilities</td>
<td>description of feature types operations supported on each feature type</td>
</tr>
<tr>
<td>describefeaturetype</td>
<td>description of the structure of a feature type</td>
</tr>
<tr>
<td>getfeature</td>
<td>retrieve feature instances respecting spatial and non-spatial constraints</td>
</tr>
</tbody>
</table>

Table 4.5: Description of existing WFS 1.0.0 request types.

When created, the exposed FeatureTypes (auto-generated upon user/campaign creation or not) can be customized by a site administrator. In figure 4.10 we show the control panel for a FeatureType auto-generated for a User. We can see that an administrator can customize which properties are exposed, make custom queries for the exposed objects and add more information for the FeatureType.

![Change feature type interface](image)

Figure 4.10: Administration panel for the WFS FeatureType associated with a SandDB User.

Querying the WFS GetFeature endpoint for a user, the valid XML response is as show in listing 4.8. Theoretically a user wouldn't ever be exposed to this XML though because this would be consumed by...
a GIS system such as QGIS. One can also load this data into QGIS and explore the data as shown in figures 4.11.

Listing 4.8: WFS GetFeature. Truncated for brevity.

(a) Add a WFS layer in QGIS pointing at the WFS endpoint exposed by Django-WFS.

(b) The loaded data with exposed attribute table.

Figure 4.11: How to use the WFS service with QGIS.
4.6 User Interface and User Experience

In this section we describe external dependencies of the front-end, with a quick overview of used technologies. After that we will go through the different interfaces and show how the User Experience of SandDB was designed.

4.6.1 External dependencies

To build the front-end we used some existing frameworks to help speed up development. To ease the interaction with the DOM in a browser compatible way we used the jQuery Javascript framework. To get data to the user we rely on both the Django templating engine and jQuery for asynchronous requests. In order to display georeferenced data in the browser we use a mobile-friendly interactive mapping library, Leaflet, backed by tiles from OpenStreetMap which could be easily swapped for Google Maps, for example. For plotting on the browser we currently use Flot, a library based on jQuery. To help with data importing and exporting visually in the browser we used handsontable which a Javascript library to emulate a spreadsheet in the browser.

Finally, to help with the design, appearance and usability we based the HTML markup on Bootstrap which has support for responsive layouts, custom HTML and CSS components and some helpful jQuery plugins.

4.6.2 User registration flow

User account creation, sign in and password recovery have a particular implemented flow which is demonstrated in figures 4.12 and 4.13. The flow also supports the usual user email confirmation flow where before logging in a user, an email is sent to the address supplied with a unique token. When the URL with the token is accessed the account is activated and only then can the user log in. These steps of account activation can be controlled by a site admin.

* username
* email address
* password
* password confirmation

![User registration flow diagram](image)

Figure 4.12: User registration flow.
4.6.3 Importing and Exporting data

Importing data into SandDB was a main concern of ours when developing the platform. Even though we have built an API and regular browser forms to import data, each of them has a problem. The API will only be used by advanced users which integrate some tool they already use. On the other hand, the forms, while perfectly good for a small amount of information, quickly become an herculean effort if we imagine data sets with hundreds or thousands of samples. As such, we devised other methods of importing data into the platform.

The first is a bulk import form which allows for faster data entry of multiple samples. It expands to insert as many samples as we want at the same time. This method is shown in figure 4.14.

---

**Figure 4.13: User sign in flow.**

**Figure 4.14: Bulk import of data through a custom form.**
The second bulk import builds upon the first and can be accessed by pressing "switch to spreadsheet view" in the previous screen. It emulates a spreadsheet in the browser and is compatible with tools such as Excel regarding copying and pasting cells and formatting both into and out of the browser. It also allows for any number of samples to be inserted as the spreadsheet auto-expands with insertion. This interface is shown in figure 4.15.

![Spreadsheet interface in the browser.](image)

The final bulk import method works by creating (or downloading our default file) a .xls file with the appropriate columns and filling the data directly in a desktop spreadsheet program of choice. This file can then be imported into the platform and new data will be imported, as well as supporting smart updates to existing data. This flow can be seen in figures 4.16.

![Import from spreadsheet file flow.](image)

### 4.6.4 Other pages

Regarding the visual aspect of the interface, in its current form it looks as its shown in the following figures:

- Figure 4.17 shows the user Dashboard, the main page a user sees upon login. In this page a user can see his latest Campaigns and Samples and has controls to add new ones.

- Figure 4.18 shows a Campaign detail page where anyone can see a Campaign and its properties. It includes a map with all the Samples associated with the Campaign.
• Figure 4.19 shows a Sample detail page where anyone can see all the information associated with a given Sample. It shows all the analysis that were performed, the classifications that the Sample was given as well as pictures and associated PDF’s. We can also see where the Sample was collected in an interactive map.

• Figure 4.20 shows a search page. It is showing a Sample search and it shows a way to refine the search by the user which collected the Sample. These searches are powered by the external search server and can be refined by different parameters.

All of the presented data is for testing purposes only and doesn’t represent real samples.
Figure 4.18: SanddDB campaign page. Test data.
Sample vasco7

Retrieval date: Dec. 27, 2014
Retrieval location: Tonle Sap
Latitude: 13.7005
Longitude: -104.1396
Retrieval depth: 1.0 meters
Retrieval height: 2.3 meters
Location accuracy: 10.0 meters
Geodetic datum: ITRF00-1206 WGS 84

Classifications

Classification type
Chemical element: Dentium
Grain size: Argillaceous rock

Granulometry

Initial weight (g): 92.4 grams
Final weight (g): 41.9 grams
Error: -2%

<table>
<thead>
<tr>
<th>Size (µm)</th>
<th>Weight (grams)</th>
<th>Weight (%)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-11</td>
<td>5.90</td>
<td>12.20</td>
<td>Very large boulders</td>
</tr>
<tr>
<td>-10</td>
<td>4.00</td>
<td>9.10</td>
<td>Large boulders</td>
</tr>
<tr>
<td>-9</td>
<td>9.00</td>
<td>18.37</td>
<td>Medium boulders</td>
</tr>
<tr>
<td>-8</td>
<td>7.00</td>
<td>14.29</td>
<td>Small boulders</td>
</tr>
</tbody>
</table>

pH

pH value: 3
Classification: Acidic

Photographs

NS-FUJ-CCARSE-SAN...
Super sandy sand

Other data

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>holes_11_Sheet.pdf</td>
<td>No description provided.</td>
<td>22.0 KB</td>
</tr>
<tr>
<td>All_samples_11_Sheet_patches.pdf</td>
<td>No description provided.</td>
<td>384.3 KB</td>
</tr>
</tbody>
</table>
Figure 4.20: SandDB filtering samples by classification type. Test data.
4.6.5 Deployment

We predict three possible deployment cases for this thesis’ project, consisting of private and/or public nodes. Private nodes would satisfy institutions for which data privacy is the most important requirement. Public nodes would satisfy our requirements of discoverability and community participation. A hybrid solution would allow users of private nodes to dynamically share partial datasets with some set of other nodes in the network. This hybrid solution is out of the scope of this project because its dynamic permission models would have to grow from a more stable platform. The three possible types of deployed networks are shown below:

1. Single node with several institutions and users.

2. One private node per institution, one node for public users.

3. A hybrid federated network of public and private nodes.
5. Conclusions

In this thesis the problems geology professors, students, researchers and enthusiasts face with sample archival and discovery were presented. We analyzed existing tools which tackle this problem and found them inadequate for their broad scope, for not allowing everyone to participate and for being hard to use. We defined some subjective goals of a platform which could eliminate these problems and turned them into technical requirements. A lot of time was dedicated to the comparisons of existing supporting tools, frameworks and libraries that would be a good fit for development and future maintainability of the project.

The SandDB platform is a first step to tackle a bigger problem of collaboration and sharing of scientific data, in the specific case of sediment samples. It proves it is feasible to implement a web based GIS system using open-source technologies and web frameworks. This is currently not obvious to many people who work with these types of systems, specially when mostly exposed to proprietary solutions. The open-source ecosystem around geospatial data is in fact very mature and ready to be used by this genre of applications.

In a more broad scope, the implementation of the WFS standard into Django can be a first step to further implementations of other OGC standards and allowing for the creation of a full OGC compliant stack on top of Python and Django without requiring a full-fledged external server as a dependency for geospatial projects. This implementation is specially important to make evident that going from official specification to working software isn't that hard. In some cases blindly using full blown external dependencies for these services can be too much.

While proving that implementing a web-based GIS system in a short period of time is possible, it would be desirable to test it with more extensive (and real) datasets. Some efforts were made in order to obtain collaboration but these weren't immediately successful, which doesn't mean future attempts can't be.

While most of the thesis is aimed at the particular case of sediment samples, by using the core of the work - REST API, User management, front-end mapping capabilities, Django-WFS - one has most of the pieces all ready to develop other similar GIS systems.

5.1 Future Work

From this first stage of research and development we have created a base system which supports most of the functionalities we believe are needed for the tool we proposed to create. As for other future work that we anticipate:

1. Implement more analysis as users require them
2. Implement a more fine-grained permission system
3. Implement user interaction directly into the platform in the form of Sample requests.
4. Increase discoverability in the form of map based interactive searching.

Another non-technical goal can also be defined, which will nonetheless arguably be of great importance to the viability of this work, getting users. By getting users into the development loop, providing feedback and using the platform, bugs will be more rapidly discovered, most used features will be identified and improved upon and unnecessary features might be discarded.
Bibliography


