POLITECNICO DI MILANO

Scuola di Ingegneria Civile, Ambientale e Territoriale

Master of Science in
Geoinformatics Engineering

Landslides Monitoring Geographic Information System

Supervisor:

Prof. Maria Antonia Brovelli

Master thesis of:

Edoardo Pessina Matr. 905514

Academic Year 2018 – 2019

Politecnico di Milano:
www.polimi.it

Scuola di Ingegneria Civile, Ambientale e Territoriale:
www.ingcat.polimi.it

Alla professoressa Maria Antonia Brovelli,
per aver creduto in me.

Contents

List of Figures

List of Listings

Abstract

Landslides are one of the most common natural hazard phenomena and represent a serious safety problem both for the environment and for the anthropic activities. The goal of project MHYCONOS consists in giving a response to the public need for security, raising awareness on the potential risk of landslides, especially in mountain areas. In order to fulfill this purpose, the project implements a parsimonious (i.e. with few parameters) model of shallow landslides initiation, explicitly taking in consideration the effects of rainfall, snow melt and climate changes. The results of project MHYCONOS are landslide susceptibility thresholds from which it is possible to create susceptibility maps for zones prone to instability.

An effective communication of these results to the public could prevent fatalities and damages and could lead to the development of mitigation measures where the land is susceptible to instability. To accomplish this goal, the project needs a precise and consistent geographic information system. This thesis exposes the work done to design and implement such a system.

The core of project MHYCONOS GIS is composed of a spatial database built to store the inputs and the outputs of the project itself, and of a WebGIS which presents these data in a simple, coherent and interactive way. Moreover, a mobile application enables common citizens to contribute to the project gathering data about landslides on the field. These entries can be explored, filtered and downloaded through a QGIS plugin.

Keywords: Geographic Information System, project MHYCONOS, landslides, free and open source software, Tartano basin.

Sommario

Le frane sono una delle più comuni calamità naturali e rappresentano un serio rischio sia per l'ambiente sia per le attività antropiche. Il progetto MHYCONOS si pone l'obiettivo di dare risposta alla necessità pubblica di sicurezza, incrementando la consapevolezza del rischio potenziale delle frane, specialmente nelle aree montuose. Al fine di realizzare questo scopo, il progetto implementa un modello parsimonioso (ovvero con pochi parametri) di innesco di frane superficiali, tenendo in considerazione esplicitamente gli effetti di pioggia, scioglimento nivale e cambiamento climatico. I risultati del progetto MHYCONOS sono soglie di suscettibilità da frana, dalle quali è possibile creare mappe di suscettibilità per le zone soggette a instabilità.

Una comunicazione efficace di questi risultati al pubblico potrebbe prevenire vittime e danni nonché condurre allo sviluppo di misure di mitigazione nei luoghi in cui il terreno è suscettibile a instabilità. Per raggiungere questo obiettivo, il progetto necessita di un sistema informativo territoriale preciso e consistente. Questo elaborato di tesi espone il lavoro svolto per progettare e implementare tale sistema.

Il nucleo del SIT del progetto MHYCONOS è composto da un database spaziale, costruito per memorizzare gli input e gli output del progetto stesso, e da un WebGIS che presenta questi dati in modo semplice, coerente e interattivo. Inoltre, un'applicazione per telefoni cellulari consente ai comuni cittadini di contribuire al progetto raccogliendo sul campo dati relativi alle frane. Questi dati possono essere esplorati, filtrati e scaricati mediante un plugin di QGIS.

Parole chiave: Sistema Informativo Territoriale, progetto MHYCONOS, frane, software libero, Val Tartano.

Chapter 1

Introduction

Landslides represent one of the major natural hazard phenomena spread all over the globe in different forms and geographical contexts.

In the Lombardy region, 130.450 landslides have been recorded, covering an area of 3.300 square kilometres correspondent to about 7.2% of the region extent [Ceriani and Fossati, 2006]. 41% of these phenomena are classified as *shallow landslides*, events consisting in the sliding of a mass of granular material on a shallow, planar surface parallel to the ground.

A considerable number of shallow landslides is caused by infrequent meteorological events which disrupt the delicate equilibrium between the soil resistance to failure and the gravitational forces acting downslope. Liquid water (e.g. rainfall and snow melt) is a recognized triggering factor of shallow landslides; therefore, scientists have been focusing on the task of defining the rainfall amount needed for slope failure initiation for the last decades.

It is precisely in this context that the *Fondazione Cariplo* project called *MHYCONOS* - an acronym for "Mapping the HYdrological Control on shallOw landSliding" - finds its collocation. Project MHYCONOS aims at the creation of a robust methodology to determine the hydrological thresholds above which shallow landslides are triggered, taking into consideration the joint action of rainfall and snow melt.

This thesis retraces the work done to design and develop the informatics and geoinformatics tools required for the project.

Section 1.1 provides a general description of project MHYCONOS and of its case study. In Section 1.2 the requirements for each needed software are outlined. Section 1.3 is a study on the current state of the art and Section 1.4 presents the general outline of the thesis.

1.1 Project MHYCONOS

Nowadays we are witnessing a growth in the importance of proper land management, with a strong focus on the areas characterizes by a rapid urbanization process.

In this context, it has been observed that the Lombardy region is heavily affected by hydro-geological natural hazards. This is particularly true in pre-Alpine and Alpine areas, where rainfall intensity and duration combined with snow melt often set hydrogeological conditions favourable to the occurrence of shallow landslides.

The contribution of snow melt in the triggering process of that kind of hazards actually tends not to be commonly investigated. Neglecting this aspect of the

problem, however, could result in an underestimation of shallow landslides risk, especially in mountain regions where spring thaw tends to be the cause of a large share of mass movements. The lack of appropriate prediction, prevention and mitigation measures leaves the population exposed to such disasters.

There is therefore a pressing need to help society to understand how and in what conditions a storm event makes a site susceptible to shallow landslides. It is for this reason that project MHYCONOS was initiated with the goal of answering to the public need for security.

In particular, the project is framed around four main objectives:

1. developing a systematic framework for the analysis of the rainfall thresholds of slope instability;
2. gathering data on soil and on hydrological conditions both on the field and in laboratory;
3. studying shallow landslides triggering from the point of view of transient and future climate change;
4. designing and implementing of a GIS database to store and handle the main outputs and present them in an interactive and appealing way.

The ultimate goal of project MHYCONOS is the creation of maps reporting landslides susceptibility under present weather state and climate change scenarios.

The project is a joint effort of *Politecnico di Milano* and *Università degli Studi di Modena e Reggio Emilia*.

1.1.1 Case study

As already stated, the Lombardy region is particularly affected by shallow landslides, particularly in the pre-Alpine and Alpine areas, where mass movements and flash floods are common.

Hence, the case study of the project is an area paradigmatic of this phenomenon in the Alps of Lombardia, the *Tartano basin*. This valley is located in the northern part of Lombardy in the province of Sondrio (see figure 1.1). The basin crosses the municipalities of Tartano and Talamona, which are involved in the project together with the province of Sondrio as stakeholders and end-users.

The choice of the Tartano basin as case study is due to the particularly high number of shallow landslides which occur in that region, especially during the snow melt season. Moreover, in 1987 a large shallow landslide event occurred in the area with severe consequences: a residential building and a resort hotel were destroyed causing the death of 14 people.

For this reason, population and stakeholders in this zone are particularly interested in tools that may advance knowledge of the hydrogeological risk, reducing the chance of accidents and casualties and improving the quality of life.

1.2 Software requirements

One of the main goals of project MHYCONOS is to expose the outcomes of the work in a clear, consistent and understandable way. This can be achieved through

Figure 1.1: Boundaries of the Tartano basin (in red).

a *geographic information system* (GIS).

The work detailed in this thesis has the purpose of creating the aforesaid information system, providing the research team with a series of software capable of gathering, storing and displaying the data and the results of project MHYCONOS.

The first requirement concerns the design and implementation of an efficient and user-friendly *GIS database* able to store all the information required by the project partners, from the spatial data regarding the case study area to the mathematical model inputs and to the results.

Those data have to be displayed in a way that can be easily disseminated and communicated to the public. Hence, the second requirement is the creation of a *WebGIS* built accordingly to GIS-compliant standards in order to present the work in an interactive and appealing way.

The third requirement moves the focus on the possibility to enhance the involvement of the citizens in the project engaging them in the data gathering process. To achieve this goal, a *mobile application* has to be built with the purpose of providing to professionals and non-professionals a tool for mapping landslides on the field in an easy and guided way.

The last requirement consists in the creation of a QGIS *plugin* capable of retrieving, filtering and displaying the data collected through the previously mentioned mobile application.

Every piece of software must be optimized to deal with spatial data and to be as consistent and easily understandable as possible. Moreover, the code has to be free and open source and compliant with GIS open standards.

1.3 State of the art

The software developed for this thesis have been specifically designed for project MHYCONOS following the requirements of the project partners. It is therefore fairly difficult to find works with comparable characteristics.

The exploitation of mobile applications for the collection on the field of data about landslides, however, is explored in various projects and literary works, usually in combination with a WebGIS which makes it possible to visualize the results.

Some publications keep this concept just at a theoretical level. For example, [Rosser et al., 2017] presents the development of an application for landslides reporting as a future improvement for the New Zealand national landslide database. A similar suggestion can be found in [Kreuzer et al., 2017], where the creation of mobile GIS applications capable of collecting data during field work is theorized in the context of the development of a new national landslide database for Germany.

On the other hand, other projects actually implement functioning applications for the gathering of data on different scales.

One example is project *COOLR* (Cooperative Open Online Landslide Repository) developed by NASA in 2007 with the purpose of creating a worldwide inventory of landslide events exploiting citizen science [Juang et al., 2019]. The project consists of a Web-based application called Landslide Reporter where scientists and citizen scientists can submit landslide reports. All the data collected are openly available through the Landslide Viewer WebGIS.

On a smaller scale, in response to the catastrophic cyclone Tamara, which stroke Serbia and the surrounding countries in May 2014, the *BEWARE* (BEyond landslide aWAREness) project started from the joint effort between the Geological Survey of Serbia and the University of Belgrade Faculty of Mining and Geology. The scope of the project was the creation of a standardized landslide database and the involvement in the process of the 27 Serbian municipalities affected by the cyclone [Mikoš et al., 2017]. To achieve these results, the project team developed an Android application for interactive landslide events reporting and a WebGIS to visualize, query and download the data.

Another honorable mention is the work of the University of Lausanne that, together with the Tribhuvan University in Nepal, implemented a mobile-map application called *ROOMA* (Rapid Offline–Online Mapping Application) designed for a rapid offline collection of landslide hazards and risk data [Olyazadeh et al., 2017]. The application allows users to locate themselves on a map and to collect data on landslide events, hazard impacts, and damaged infrastructure. If the Internet connection is available, the entries are then uploaded to a server.

1.4 Thesis outline

The thesis is structured in the following way:

Chapter 1 introduces the work and the current state of the art.

Chapter 2 describes the design and the implementation of the GIS database. This chapter provides details about the structure of the database and about

the tools utilized in the process. Moreover, it presents some details on the exploration and preparation of the spatial data currently in the database.

Chapter 3 illustrates the characteristics and the functionalities of the WebGIS. On top of that, it contains a digression on the main standards implemented by the software.

Chapter 4 deals with the design and the development of LandslidesSurvey mobile application. This chapter presents the features of the application together with its architecture and some details about its code and documentation.

Chapter 5 contains an in-depth description of the testing process of LandslidesSurvey mobile application. Firstly, the chapter gives a general overview on the development and the testing plan of the software. After that, it goes into details about the various testing levels applied to the product.

Chapter 6 presents the development process of LandslidesSurvey QGIS plugin focusing on the features and on the implementation details of the program.

Chapter 7 concludes the thesis and exposes possible future developments of the various implemented software.

Chapter 2

Data

One of the key aspects of the project is the design and the implementation of an efficient *GIS database* capable of housing multiple spatial data gathered with different methods.

In this phase, particular attention has to be put on the user friendliness of the final product, since one of the main purposes of the database is to expose the model results in a manner which can be used for rising societal awareness on shallow landslide hazard.

Generally speaking, every information required by the project partners - both for the study of the area of interest and as inputs of the model itself - has to be rationalized and stored in the database. On top of that, the design of the system has to be compliant with the various outcomes of the project, whether they are the outputs of the model or any auxiliary data produced during the work period.

Section 2.1 addresses the details of the exploration and preparation of the data. Section 2.2 gives an overview on how the database is structured and on the tools used in the project and section 2.3 dives into details about the specific spatial data stored in the database.

2.1 Data preparation

In order to help contextualizing the project and the model outputs, the database must contain first and foremost all the spatial data that could be useful to explore the area of interest (namely the Tartano basin and its surroundings). The topology of the zone, its hydrological characteristics, and the anthropic elements are example of valuable information that has to be stored in the database.

Before doing this, however, those data have to be collected, explored and processed to verify their integrity and to remove redundant or unnecessary features. In the context of the project, this task was accomplished using *QGIS Desktop*, a free and open-source geographic information system application that allows the visualization, managing, editing and analysing of geospatial data.

Generally speaking, the graphical representation of spatial data can take the form of either *vector* or *raster* data. The difference is that vector graphics are rendered through a mathematical manipulation referenced by coordinates, while raster graphics use pixels arranged in a grid-type architecture to store the data. In other words, vector data are mainly used to represent discrete spatial features, made of georeferenced points, lines or polygons, while rasters are more suitable for storing data that vary continuously in space, like aerial photographs or satellite

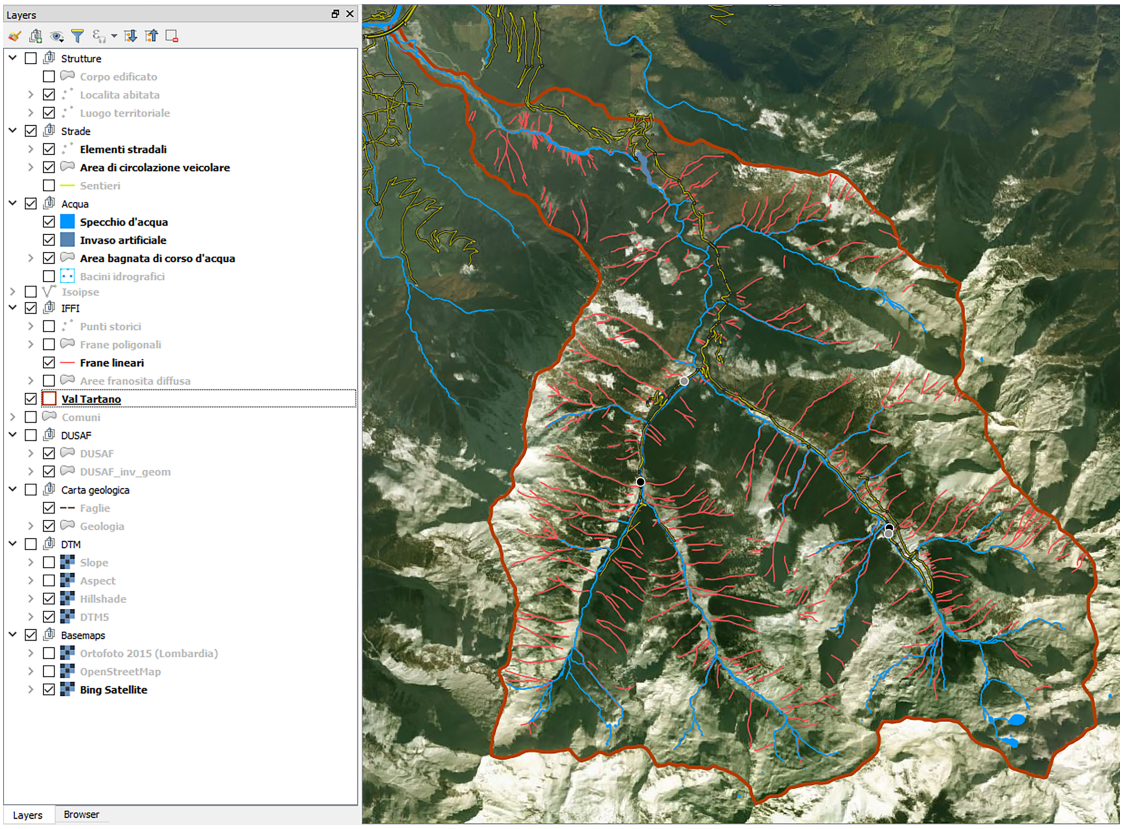


Figure 2.1: QGIS project showing the spatial data currently in the database.

images.

Figure 2.1 shows an overview of all the layers - vectors and rasters - that compose the database rendered and styled in QGIS.

Leveraging QGIS powerful editing tools, each of the layers has been subjected to the series of processes detailed below with the purpose of preparing the data to be stored in the database.

Reprojection

The first task is to ensure that each layer has the same coordinate reference system (CRS). A CRS is a system "for uniquely referencing spatial information in space as a set of coordinates [...] based on a geodetic horizontal and vertical datum" [Infrastructure for Spatial Information in Europe, 2019].

Even if QGIS implements a feature called *on the fly reprojection* which allows layers to be automatically projected in the CRS of choice, it is important that the data in the database are actually saved with the same CRS.

In particular, the coordinate system chosen for the project is *WGS 84 / UTM zone 32N* (EPSG:32632), the official Italian reference system.

Clipping

Almost every layer has an extent far greater than the actual area of interest. Saving those extra features in the database would just be a waste of space, slowing the

computations and making the visualization less clear.

For this reason, each layer (whether vector or raster) is subjected to an operation called *clipping*. This process allows the extraction of all the features of a dataset that fall within the polygons of a certain overlay layer. Using the extent of the Tartano basin as the boundary, the resulting layers contain only the features of interest for the project.

Styling

A clear and careful styling process is key for a meaningful and easily readable map.

QGIS offers a great tool for managing the symbology of layers with the possibility of styling features according to custom rules and defining the rendering of raster bands.

Moreover, the styles defined in QGIS can be exported as `.sld` files. The Styled Layer Descriptor is a standard to describe the appearance of map layers defined by the *Open Geospatial Consortium*, an international organization committed to the creation of quality open standards for the global geospatial community. In practice, it defines an XML schema that can be used to encode the style of a layer.

Being able to store this file in the database guarantees the visual consistency of the map independently from the software used to render it.

Metadata definition

The last step of the process consists in gathering and specifying the *metadata* of each layer. Metadata are data that provides information about other data and are extremely important in order to contextualizing the dataset.

In particular, language, brief description, utility company, data of creation and license are specified for each layer title.

2.2 Database implementation

A *database* is a collection of data and information organized to be easily accessed, managed and updated. In the field of Geographical Information Systems, databases are typically used for the management of attributes linked directly or indirectly to geometric primitives.

Data are considered a component of primary importance in any GIS and this leads to the necessity of choosing carefully the systems and the techniques used to handle and manipulate the information they carry.

The system that allows the management of a database is called *Database Management System* (DBMS) and it is described as a "software system that enables users to define, create, maintain and control access to the database" [Connolly and Begg, 2005].

The software chosen for the project is *PostgreSQL*, a widely used free and open-source DBMS. PostgreSQL is a relational DBMS: it stores data as rows of structured *tables* where each column represents an attribute. PostgreSQL empathizes extensibility and standards compliance as well as reliability and concurrency.

However, PostgreSQL on its own is not able to deal with spatial data. It needs an extension called *PostGIS*, an open-source software that adds support for geographic objects allowing location queries to be run in SQL. Practically speaking, PostGIS introduces extra types (like geometry, geography or raster) to the database as well as predicates and operations to deal with geometry measurement and interaction.

As to now, the database is composed by one table for each of the layers (both vectors and rasters) needed for the exploration and the contextualization of the area or interest. On top of that, it is possible to find in the database two tables containing respectively the default styles and the metadata of all the layers.

2.3 Data overview

As already stated, the spatial data currently in the database cover all the information about the area of interest which may be useful to the project partners.

The entirety of those data is free and open source, available on the web and distributed by *Geoportale della Regione Lombardia* under IODL 2.0 licence[1] or by *OpenStreetMap* under ODbL licence[2].

Table 2.1 presents an overview of all the layers that compose the database with their name in English and Italian, their type (vector or raster) and the year of data collection.

2.3.1 Vector data

In the database, vector data are used to represent discrete spatial features leveraging the *shapefile format*, an Esri data format used to store location, shape and attributes of geographic features.

All the layers currently in the database are presented below.

Tartano basin (Val Tartano)

This layer contains the boundaries of the area of interest: the Tartano basin. This layer is the one used as overlay while clipping the other datasets.

Municipalities (comuni)

The dataset showing the administrative boundaries of the three municipalities in which the Tartano basin falls: Tartano, Talamona and Forcola.

Geological map (carta geologica)

A collection of data regarding lithology, fault lines, and over-thrusts of the area at a scale of 1:250.000. It is composed of the following layers:

[1] *Italian Open Data License*. It allows the user to use, modify and share the data with the obligation to cite the original source.

[2] *Open Database License*. It allows the user to share, create and adapt the data under the conditions of attributing, sharing-alike and keeping open any work.

Table 2.1: Summary table of all the spatial data in the database.

English name	Italian name	Data type	Creation year
Tartano basin	Val Tartano	Vector	2017
Municipalities	Comuni	Vector	2018
Fault line	Faglie	Vector	1990
Geology	Geologia	Vector	1990
Agricultural and forest land use	Destinazione d'uso dei suoli agricoli e forestali	Vector	2015
Widespread landslides areas	Aree franosità diffusa	Vector	2014
Linear landslides	Frane lineari	Vector	2014
Polygonal landslides	Frane poligonali	Vector	2014
Historic points	Punti storici	Vector	2014
Contour map	Isoipse	Vector	2008
Drainage basins	Bacini idrografici	Vector	2006
Streams wet area	Area bagnata di corso d'acqua	Vector	2008
Artificial lake	Invaso artificiale	Vector	2008
Body of water	Specchio d'acqua	Vector	2008
Vehicular circulation area	Area di circolazione veicolare	Vector	2008
Road elements	Elementi stradali	Vector	2008
Paths	Sentieri	Vector	2008
Spatial location	Luogo territoriale	Vector	2008
Populated area	Località abitata	Vector	2008
Building	Corpo edificato	Vector	2008
DTM	DTM	Raster	2015
Hillshade	Hillshade	Raster	2015
Aspect	Aspect	Raster	2015
Slope	Slope	Raster	2015

- *fault lines (faglie)*: fault lines;
- *geology (geologia)*: lithology classification.

Agricultural and forest land use (destinazione d'uso dei suoli agricoli e forestali)

This layer classifies the territory with respect to the main forms of land use and cover.

Lombardy landslides collection (inventario fenomeni franosi Lombardia)

The landslides inventory of Lombardy region. It is composed of the following layers:

- *widespread landslides areas (aree franosità diffusa)*: areas subject to widespread landslides;
- *linear landslides (frane lineari)*: landslides whose length is far greater than their width;
- *polygonal landslides (frane poligonali)*: landslides with a significant surface at a scale of 1:25.000;
- *historic points (punti storici)*: historic events derived from the information gathered in city records, press reports, existing publications, and old inspections.

Contour map (isoipse)

This layer is composed by the collection of curves that join points having the same altitude.

Drainage basins (bacini idrografici)

The dataset displaying the three drainage basins that fall in the Tartano basin.

Streams wet area (area bagnata di corso d'acqua)

This layer includes areas which contain water and are part of stream beds. Those areas are continuous and fictitiously closed by an intersection with another body of water.

Artificial lake (invaso artificiale)

This layer shows all the artificial reservoirs created using a dam or generated by productive activities.

Body of water (specchio d'acqua)

This dataset is composed by surfaces covered by still waters (e.g. lakes, ponds, swamps, lagoons).

Vehicular circulation area (area di circolazione veicolare)

This layer shows the areas where the transit and the parking of vehicles is allowed.

Road elements (elementi stradali)

Collection of parking spots and barriers in the area.

Paths (sentieri)

This layer shows all the marked paths in the valley.

Spatial location (luogo territoriale)

This dataset contains and defines the various spatial locations and objects.

Populated area (località abitata)

The layer showing all the populated areas in the valley.

Building (corpo edificato)

This layer contains the buildings in the area defined by their footprint.

2.3.2 Raster data

The main raster which is present in the database is the 2015 **Digital Terrain Model (DTM)** of the Tartano basin with resolution of 5x5 meters. This model is a three dimensional representation of the terrain surface created from the elevation data.

Starting from the DTM, the following three raster layers have been calculated:

- **hillshade**: a gray-scale 3D representation of the surface, where the value of each cell is the illumination caused by a given source;
- **aspect**: a representation that identifies the down-slope direction of each of the cell faces;
- **slope**: a map that represents the maximum rate of elevation change between each cell of the DTM.

All those rasters are provided in the *GeoTIFF* format, a standard image file format that embeds additional spatial information in the file, such as the spatial extent, the coordinate reference system, and the resolution.

Chapter 3

WebGIS

As already stated in Chapter 2, QGIS is a totally viable option to visualize the layers into the database. Offering the possibility to seamlessly connect to PostgreSQL, QGIS enables users to query and edit the datasets directly from their machines. Moreover, it is possible to grant write privileges only to a limited number of users leveraging on the authentication system implemented by PostgreSQL to avoid any unwanted modification.

This approach, however, has two main disadvantages. First of all, QGIS must be actually installed on the computer and secondly, the program still needs a certain degree of working knowledge to be manoeuvred despite its user-friendliness. For these reasons, amateur users, who just want to browse through the results of the project but have not got any knowledge or possibility to use QGIS, may be discouraged.

The solution for this problem came in the form of a *WebGIS*, a more accessible, affordable, and pervasive way of sharing project MHYCONOS data. The source code of the software can be found on GitHub.[1]

Section 3.1 provides an overview on the general characteristics of a WebGIS and of the standards it implements. In Section 3.2 the main functionalities of MHYCONOS WebGIS are outlined. Lastly, section 3.3 presents the high level architecture of the system as well as its main components and their interaction.

3.1 General characteristics

A WebGIS is a "type of distributed information system, comprising at least a server and a client, where the server is a GIS server and the client is a Web browser" [Environmental System Research Institute, 2019]. In other words, in a WebGIS the geospatial data exposed by a server are distributed through the Web and consequently are freed from the machine that actually contains the data.

This approach guarantees several benefits:

- the data can be updated easily and on-the-fly;
- a WebGIS can reach a wide and distributed audience;
- the Web allows the exposure of the data in an interactive way, with a multimedia and dynamic presentation;
- data exchange is regulated by well-known and established standards;

[1] https://github.com/epessina/mhyconos-webgis.

- traveling on the Web, data are potentially independent from the platform used to visualize them.

3.1.1 Interoperability

Interoperability is arguably the key requirement for a seamless exchange of spatial data over the Web. Interoperability is the "capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units" [International Organization for Standardization, 2015].

A WebGIS achieves this fundamental feature through *standardization*, following in particular the open standards[2] set forth by the Open Geospatial Consortium (OGC).

In particular, the *OGC Web Services* (OWS) are services developed to expose geographical functionality to the Web through standard Web protocols. They include services for data access, data display and data processing. Their functioning can be described in four steps:

1. a client reaches a server to ask for the functionalities it implements;
2. the server responds with an `.xml` file containing these functionalities;
3. the client requests the needed data from the server;
4. the server provides the requested data.

The most widely known OWS implementations include the *Web Map Service* (WMS), the *Web Feature Service* (WFS), and the *Web Coverage Service* (WCS).

Web Map Service

WMS lets the user generate maps from geographic information dynamically, making them available as georeferenced images (i.e. rasters).

A WMS request specifies which geographic layers have to be processed over which area of interest. The response carries one or more geo-registered map images (returned for example as `.png` or `.jpeg`) that can be displayed in a browser application.

This standard defines three operations:

- **GetCapabilities**: returns service-level metadata about the available data and the accepted request parameters.
- **GetMap**: returns a map whose geographic and dimensional parameters are well-defined.
- **GetFeatureInfo**: returns information about particular features shown on a map (this operation is optional).

WMS maps can be styled using the Styled Layer Descriptor standard already presented in Section 2.1.

[2] An open standard is a standard publicly available that can be implemented on a royalty-free basis.

Web Feature Service

WFS offers a direct, fine-grained access to geographic information at the feature and feature property level. OGC establishes that features have to be encoded using either *GML* (Geography Markup Language) or KML (Keyhole Markup Language, leveraged by Google).

This standard defines eleven operations among which the most commonly used are:

- **GetCapabilities**: provides a readable description of the available features.
- **DescribeFeatureType**: a schema description of feature types.
- **GetFeature**: returns a selection of features from a data store.
- **Transaction**: allows the manipulation of the data creating, updating or deleting features in the Web feature service data store.

Web Coverage Service

WCS supports electronic retrieval of geospatial data as *coverages*, defined as digital geospatial information representing phenomena that vary in space and/or in time.

This standard defines three operations:

- **GetCapabilities**: returns metadata about the service and the offered coverages.
- **DescribeCoverage**: returns the full description of one or more coverages.
- **GetCoverage**: returns the coverage itself encoded in a well-known coverage format.

3.2 Functionalities

MHYCONOS WebGIS is built to show the data related to the project in a straightforward and user-friendly way. It is thought for amateur users, who just want to visualize the results of the work; for this reason, the user interface is kept simple, clear, and familiar.

Figure 3.1 shows the main view of the WebGIS with a numeric label placed on each of the main interface elements, which are detailed in the following sections.

(1) Map view

The majority of the screen is occupied by the map itself, where the basemaps and the layers are rendered. The map view and the zoom level can be modified using the mouse on PC and the fingers on mobile.

In the bottom right corner the user can find the buttons for zooming in and out, a scale line that shows the current scale of the map in metric units, and the attribution stating the source of the currently displayed basemap.

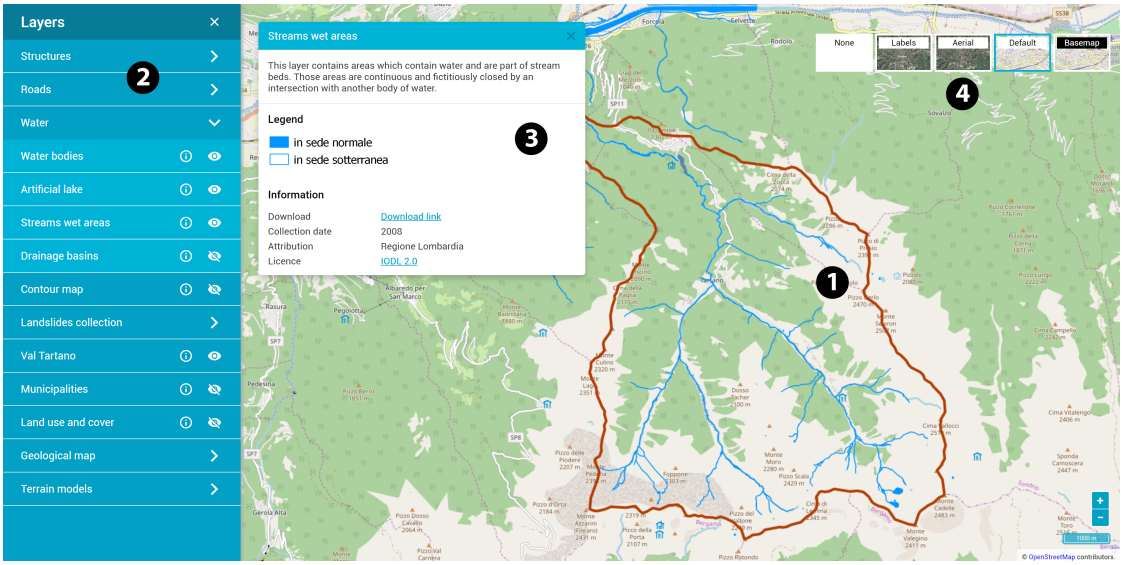


Figure 3.1: Main view of MHYCONOS WebGIS.

(2) Layers panel

The left side of the page is occupied by the layers panel. Through this menu the user can choose which layers must be shown on the map enabling or disabling them using the *eye* icon on the right of the layer name. Moreover, the *info* icon opens a small window containing some useful metadata about the layer.

To simplify the management, some layers are gathered in thematic groups that can be expanded or collapsed using the arrow on the right of each group title.

It is worth mentioning that for performance reasons only the layer containing the boundaries of the Tartano basin is loaded when the page is opened. All the other datasets are retrieved from the server on demand when the user clicks on the *eye* icon of the correspondent layer for the first time.

(3) Metadata window

This box is opened when the user clicks on the *info* icon next to the title of a layer. It contains some information about the dataset that may be useful to better understand what is being rendered.

In particular, the metadata window shows:

- the name of the layer;
- a brief description of its content;
- a legend, if needed;
- the license under which the data are distributed;
- the attribution;
- the date of creation of the dataset.

To improve the user experience, especially on small devices, this window can be dragged around using the mouse or the fingers.

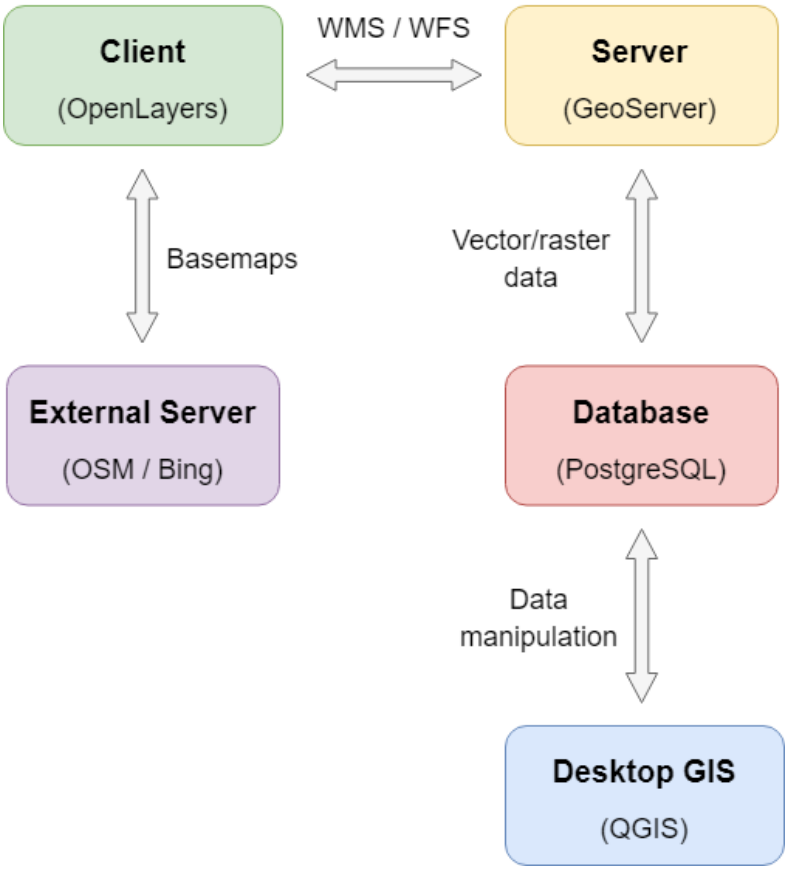


Figure 3.2: High level view of MHYCONOS WebGIS architecture.

(4) Basemaps menu

Through this menu the user can choose which basemap she/he wants to use as background for the layers. Currently, there are four options available:

- OpenStreetMap standard tile layer;
- Bing aerial map;
- Bing aerial map with labels;
- no basemap at all.

3.3 Architecture

Figure 3.2 shows a high level view of the system architecture.

At the bare minimum, Web mapping technologies require a client-side application, which renders the layers, and a server-side program that feeds the client with the needed data. MHYCONOS WebGIS implements an enhanced version of this architecture to guarantee high flexibility and a flawless user experience.

The main components of the system are detailed below.

Client

The first core component of the WebGIS is the client. It is responsible for rendering the data retrieved from the server and the external server and for implementing a well-designed interface from which the user can interact with those data.

The client of MHYCONOS WebGIS is a responsive Web page, meaning that it is built to be compatible with any screen size, from phones to large monitors. The GIS functionalities are implemented leveraging on the library *OpenLayers*.

More details can be found in Section 3.3.2.

Server

The server is the other core component of the application. All the data needed by the client have to be processed and served by the server through a series of *WMS* or *WFS* calls. In particular, MHYCONOS WebGIS is built using *GeoServer*, a free and open-source GIS server.

Section 3.3.1 presents more details about GeoServer and its characteristics.

Database & Desktop GIS

As discussed in Chapter 2, the spatial data of the project are stored inside a PostgreSQL database extended with PostGIS.

The server of the WebGIS connects directly to this database to access all the needed datasets. In this way, any change applied to the tables is reflected in real time on the server and consequently to the client.

The data in the databases can in turn be modified using a desktop GIS software like QGIS. Any change will automatically propagate all the way up to the end user.

External server

As already stated, the WebGIS gives the user the possibility to select different basemaps as background for the layers.

These maps can be retrieved through various External Servers which provide maps in form of rasters composed by *tiles*. A tile is a square bitmap graphic (usually in `.png` format) that corresponds to a particular section of a map.

This approach guarantees speed and performance since the image is broken up into tiles accordingly to the zoom level. When the map moves to a new location or to a new zoom level, the needed tiles are retrieved and the previous ones are saved in the cache of the browser.

3.3.1 GeoServer

GeoServer is a free and open-source cross-platform Web map server written in Java which enables users to view, edit, process and publish geospatial data.

GeoServer was chosen to be the core of MHYCONOS WebGIS given its maturity, flexibility and strong compliance to OGC standards. On top of that, it is built to publish spatial data from a variety of different sources, including PostGIS where MHYCONOS data are stored. Furthermore, GeoServer can be easily managed

through a browser-based administration interface that enables the configuration of every aspect of the server, from adding and publishing data to changing service settings.

For publishing data, GeoServer offers a system composed of different levels of "containers". The broader ones are the *workspaces* that are used to group similar layers together. A workspace contains one or more *stores* which represent the connection to the data sources containing geospatial data. A data source can be anything, from a file to a table in a database, and stores allow connection parameters to be defined only once, rather than for each dataset in the source. Finally, a store is materialized in the form of a *layer*, the actual raster or vector dataset that represents a collection of geographic features.

The GeoServer instance of project MHYCONOS is composed of one workspace which contains a store connected to the PostGIS database. In turn, each table of the database is published as a layer.

3.3.2 OpenLayers

OpenLayers is a free and open-source JavaScript library built to display dynamic maps in Web browsers. It provides an API which lets the developer create rich Web-based geographic applications rendering map tiles, vector data, and markers loaded from any source.

The client of MHYCONOS WebGIS relies heavily on OpenLayers to display both the spatial data hosted on GeoServer and the basemaps retrieved from the External Servers. On top of that, the library provides all the elements of the user interface needed to interact with the map, like the zoom buttons and the scale bar.

The main reason behind the choice of using OpenLayers for this project lays in its strong interconnection with GeoServer. Establishing a connection between the two entities is extremely easy since they both support all the main OGS standards.

As an example, listing 3.1 shows the function implemented to load an image through a WMS request to GeoServer.

Listing 3.1: Function to load an image through a WMS request.

```
/**
* Loads an image through a WMS request to GeoServer.
*
* @param {String} title - The title to assign to the
* image.
* @param {String} layerName - The name of the layer to
* load.
* @param {Number} [zIndex=0] - The z-index value to
* assign to the layer.
* @param {number} [maxRes] - The maximum resolution
* (exclusive) below which this layer will be visible.
*
* @returns {ol.layer.Image} The rendered image.
```

```
*/
loadWMS(title, layerName, zIndex = 0, maxRes) {

   // Configure the WMS request
   const source = new ol.source.ImageWMS({
      url         : settings.wmsUrl,
      params      : { "LAYERS": `mhyconos:${layerName}` },
      serverType  : "geoserver",
      crossOrigin: "Anonymous"
   });

   // Create the image
   const layer = new ol.layer.Image({
      title         : title,
      source        : source,
      opacity       : 1,
      minResolution: 0,
      maxResolution: maxRes,
      visible       : true
   });

   // Set the z-index to the image
   layer.setZIndex(zIndex);

   // Return the image
   return layer;

}
```

Chapter 4

LandslidesSurvey application

Nowadays, it has became clear that a precise and exhaustive landslides inventory is key to a proper susceptibility and hazard assessment. Up to now, however, this inventories appear to be rather scarce, despite the advancement of the earth observation technologies that still exhibit constrains both in time and in space. Overcoming these problems to achieve a steady supply of data about position, type and characteristics of landslides over a large space and time span could lead to a huge improvement in the proper monitoring, modelling, and mitigation of the adverse consequences of those phenomena. The ultimate goal is that of preventing casualties and damages to properties and infrastructures.

In the context of the project, it has been decided to accomplish this goal leveraging on the inclusion of non-professionals in the data harvesting process. This technique is known as *citizen science* and is defined as "the practice of public participation and collaboration in scientific research to increase scientific knowledge" [Irwin, 2002]. In other words, common citizens are asked to gather data on the field by themselves or during organized parties, using specific tools designed to ease and standardize the procedures. The massive collaboration that can occur allows investigations on a spatial and temporal scale that would never be achievable by a single scientist. A good example of what can be accomplished through citizen science is *OpenStreetMap*, a collaborative project whose aim is to create a free and editable world map.

To obtain this result, a platform for landslides surveys in the form of a cross-platform mobile application has been specifically created for the project, taking in consideration the needs of the various stakeholders. The source code of the software can be found on GitHub.[1]

Section 4.1 presents the scope and the goals of the system. In section 4.2 all the main functions of the application are detailed. Section 4.3 is an overview of the architectural design of the software, while Sections 4.4, 4.5, and 4.6 give in-depth explanations of the characteristics of client, server, and database respectively. Finally, 4.7 provides some details about the code documentation.

4.1 Scope

The scope of the *LandslidesSurvey application* is to guide users with any level of competence in mapping landslides on the field in an easy way.

[1] https://github.com/epessina/LandslidesSurvey.

The application allows users to register themselves in the system, locate themselves on a map, and tag landslides inserting information such as the type of landslide, the presence of monitoring or mitigation systems or a photo of the area through a simple or an expert interface.

All the collected data are stored in a database through a publicly open *API* that provides endpoints to retrieve, insert and modify the landslides. If no internet connection is available, the application offers the possibility of temporarily saving the data in the local memory of the phone and synchronizing them later with the server.

4.1.1 Goals

The main goals of the system are the following.

[G1] A user can register herself/himself in the system providing her/his credentials and personal information and accepting a GDPR[2].

[G2] A user can access the data regarding her/his own location.

[G3] A user can map a new landslide by specifying its position and characteristics.

[G4] A user can switch between a *simple* and an *expert* mode (see Section 4.2.3).

[G5] A user can visualize the position and the characteristics of the landslides she/he has mapped.

[G6] A user can update the data regarding the landslides she/he has mapped.

[G7] A user can delete a landslide she/he has mapped.

[G8] A user can save the data about a landslide in the local database of her/his phone if the Internet connection is not available.

[G9] A user can synchronize with the server the content of the local database of her/his phone at any time.

4.2 Main functions

Taking into consideration the goals of the application, the following sections try to give a detailed description of the main functions provided by the system.

4.2.1 Authentication

In order to have access to the full set of functionalities that the application offers, the system asks the user to authenticate herself/himself inserting valid credentials - namely an email address and a password - in the login page (figure 4.1) which is shown every time the application is lunched and no valid session is found in the local storage (see Section 4.5.2 for more details).

[2]General Data Protection Regulation.

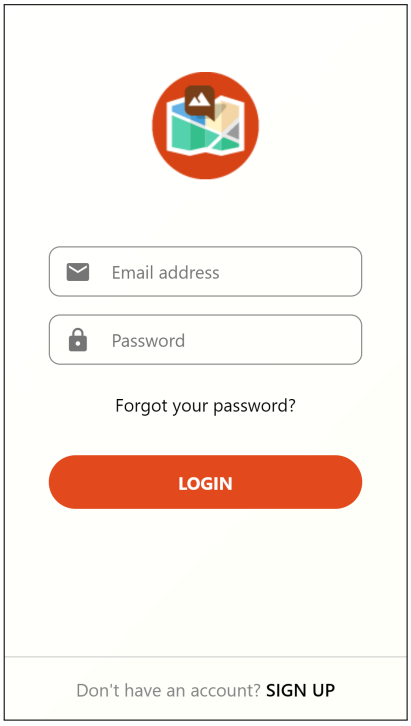


Figure 4.1: Login screen.

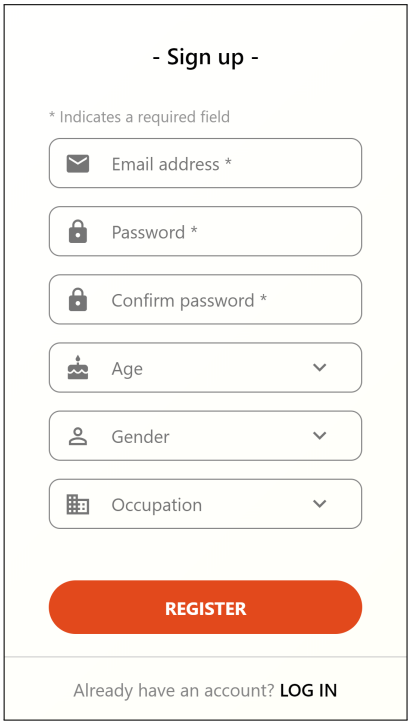


Figure 4.2: Registration screen.

If the user does not own an account, the system gives her/him the possibility to register upon acceptance of the Terms & Conditions (figure 4.2). In this phase LandslidesSurvey will request the user to provide the following mandatory data:

- a unique email address;
- a password of at least 8 characters and containing at least 1 number;

and the following optional data that can be useful for conducting statistical analysis on the usage of the application:

- range of age;
- gender;
- occupation.

In order to complete the registration process, the user is asked to confirm her/his email address through a link sent to her/his inbox. Only after this last step the user is able to login using her/his newly created credentials.

Furthermore, if the user forgets her/his password, she/he can request a *password reset* from the login page. For security reasons an email is sent to the address linked to her/his account. By clicking on the link contained in the email, the user is redirected to a Web page where she/he can choose a new password.

All of the data inserted during the registration phase (including the email and the password) can be revised by the user at any time in the settings menu.

Guest usage

Since the application is likely to be used in mountain areas or, more generally, in areas where an Internet connection could not be available, during the development of LandslidesSurvey particular attention was put on the ability to use its functionalities off-line.

For this reason, if the user starts the system without an Internet connection, it will be asked her/him to skip the login part and continue as a *guest*. In this way, she/he will be able to use only a subset of the application functionalities, namely the ones related to the local mapping of landslides (see Section 4.2.5).

4.2.2 Map

The main page of the application houses a map that displays the user's surroundings. It can be considered as the central hub of the system, since from here the user can access all the other functionalities through a set of elements displayed on top of the map itself.

Figure 4.3 shows the screen with a numeric label on each of the elements. The numbers in the following list correspond to the ones in the image.

1. **Map**. A map of the world. By default, it is centred on the current location of the user and it follows her/his movements, but its view can be manually located and zoomed. For more technical details see Section 4.4.2.
2. **User marker**. This marker shows the current position of the user. If at any time the location is not accurate, the user can drag the marker anywhere on the map to mirror her/his actual position. The coordinates that the marker has got when the *new landslide* button is pressed will be the ones of the newly inserted landslide.
3. **Remote landslide marker**. The markers of this kind show the landslides mapped by the user that are saved remotely on the main database. A click on one of those markers will open the screen showing the information about the correspondent landslide (see Section 4.2.4).
4. **Local landslide marker**. They work like the remote ones with the only difference that they represent the landslides mapped by the user that are saved in the local database (see Section 4.2.5).
5. **Settings button**. A click on this button opens the application settings (figure 4.4). From here the user can update her/his account (change information, email and password), logout of the system, change the mode of the application (see Section 4.2.3), choose the language and dive into the "help" section.
6. **Sync button**. Through this button the user can synchronize the content of the local database of her/his phone with the remote database hosted on the server (more details in Section 4.2.5). The presence of landslides to be synchronized is signalled by a red dot on the button.
7. **GPS button**. This button allows the user to center the map and the position marker on her/his location.

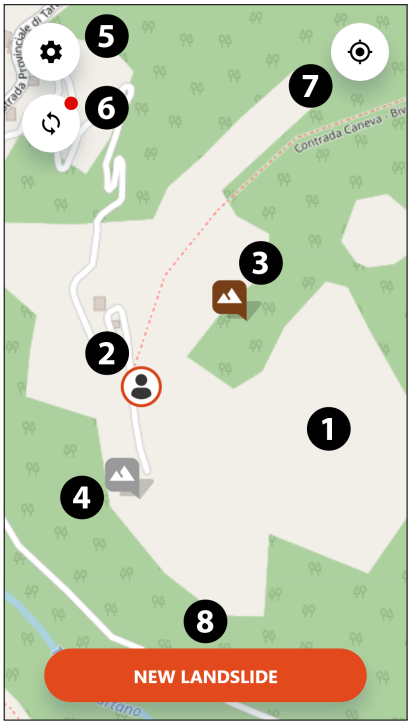


Figure 4.3: Map screen.

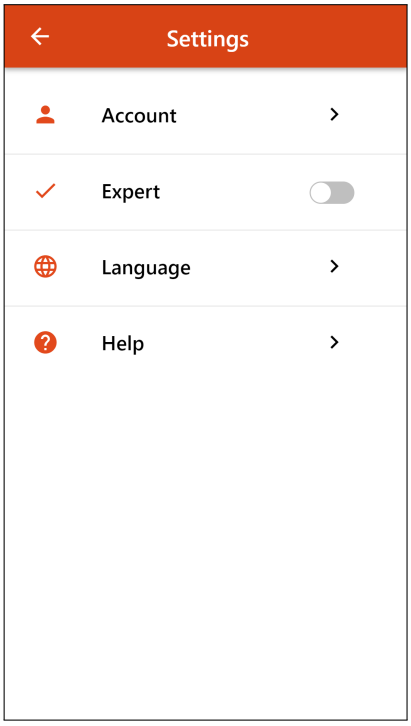


Figure 4.4: Settings screen.

8. **New landslide button**. Clicking on this button the user opens the screen to insert a new landslide into the system (see Section 4.2.3).

4.2.3 Landslides mapping

The main functionality of the application is the possibility to map landslides, recording their position (defined by the current coordinates of the user marker) and inserting useful information about their characteristics and surroundings.

The user is guided in this process by a screen of questions based on a well established gathering data flow. The application offers two different modes of operation, namely *simple* and *expert*, that are different in relation of the number and complexity of the data requested.

Simple mode

This modality is thought for amateurs who have little to no geological knowledge. Figure 4.5 shows the information requested which are detailed below.

Type of landslide Mandatory field that asks the user to identify the type of landslide. To help in the choice, the system provides a visual and written help that explains the characteristics of each type of hazard. The possible values among which the user can choose are:

- rockfall (figure 4.6a);
- toppling (figure 4.6b);

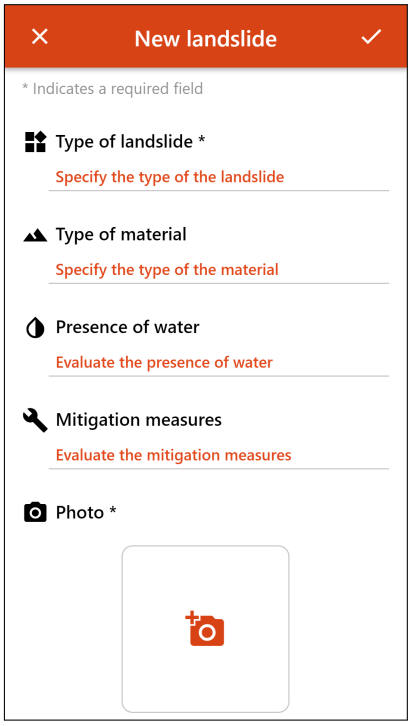


Figure 4.5: Insert screen in simple mode.

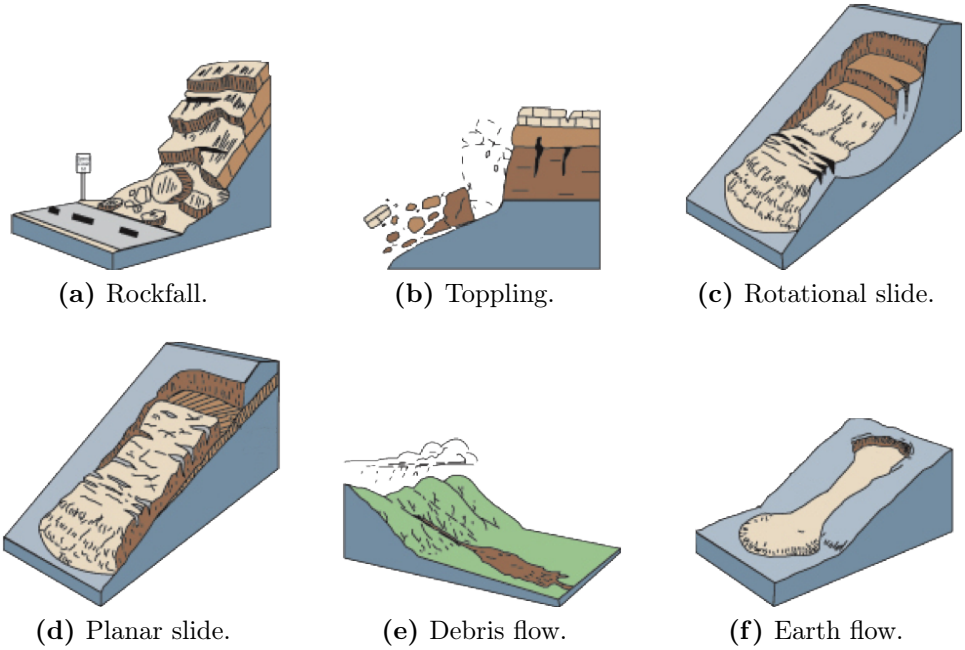
- rotational slide (figure 4.6c);
- planar slide (figure 4.6d);
- debris flow (figure 4.6e);
- earth flow (figure 4.6f);
- complex landslide (a combination of the previous types);
- other.

Type of material Optional field to specify the type of material found near the landslide. It can assume one of the following values:

- rock;
- debris;
- dirt;
- mixed;
- cannot determine.

Presence of water This optional field asks the user to qualify the presence of water in the area of the landslide. The values it can assume are:

- dry;
- humid;



(a) Rockfall. **(b)** Toppling. **(c)** Rotational slide.

(d) Planar slide. **(e)** Debris flow. **(f)** Earth flow.

Figure 4.6: Different types of landslide.

- wet;
- very wet;
- cannot determine.

Mitigation measures Through this optional field the user can assert the presence of mitigation measures in the area. It can only assume a yes or no value.

Photo Since having an actual image of the hazard is very important for any subsequent study, the user is required to take a picture of the site with her/his phone.

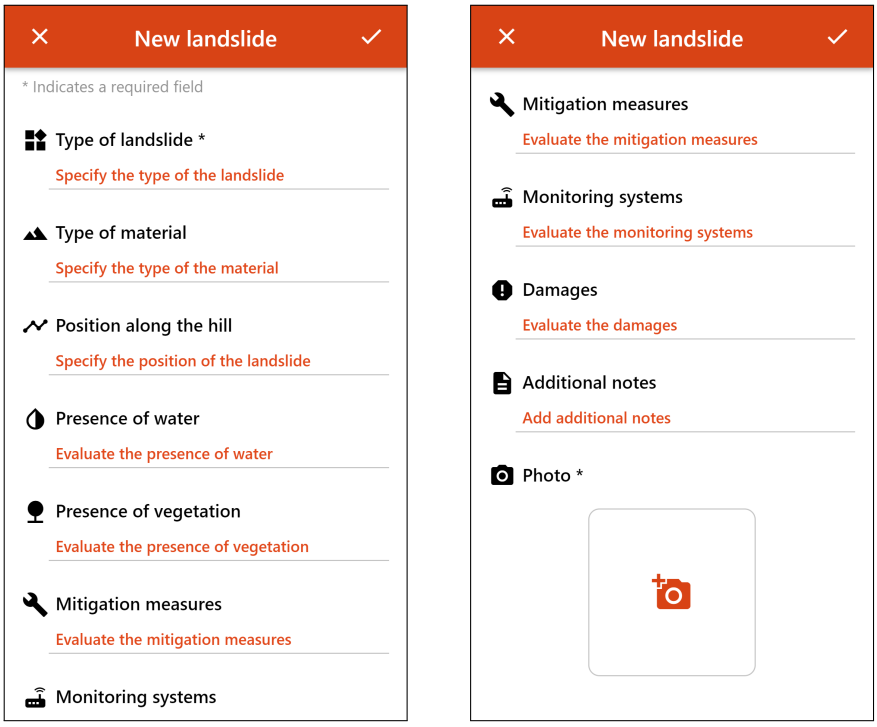
Expert mode

Users with a prior geological knowledge are advised to use the application in expert mode, since it offers a higher number of more detailed questions that allow a precise identification and description of the phenomena.

Figure 4.7 shows the insert screen in expert mode. Each information requested in simple mode is re-proposed here with various additions detailed below.

Position along the hill An optional field used to specify the position of the landslide on the hill. It can assume one of the following values:

- at the top;
- uphill;
- midslope;



(a) First part of the screen. **(b)** Second part of the screen.

Figure 4.7: Insert screen in expert mode.

- downhill;
- in the valley.

Presence of vegetation Through this optional field the user can characterize the presence of vegetation in the area nearby the landslide. The values among which she/he can choose are:

- grass;
- low-growing plants;
- trees;
- mixed;
- absent.

Mitigation measures Just as in simple mode, this field asks the user to assert the presence of mitigation measures on the site. In addition, if the answer is "yes", the application also asks to specify the type of each of the measures found, choosing among:

- retaining wall;
- gabions;
- rockfall barriers;

- spritz beton;
- friction nets;
- drains;
- geogrids.

Monitoring systems This optional field allows the user to specify if any monitoring system has been found on the spot. In case of a positive answer, the user can specify the type and the state of each system.

The available types are:

- inclinometer;
- externsometer;
- distometer;
- topographic instruments;
- GPS;
- interferometric radar;
- video surveillance.

The possible states are:

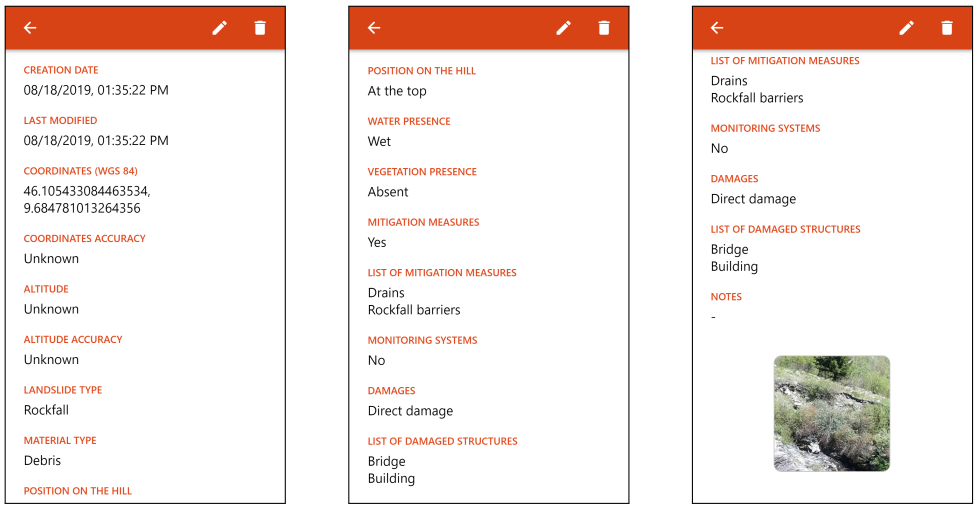
- functioning;
- partially damaged;
- damaged;
- cannot determine.

Damages Through this optional field the user can evaluate the damages caused by the landslide, choosing between:

- no damage;
- direct damage;
- obstruction of water course;
- collapsed bank or dam;
- cannot determine.

If the selected option is *direct damage*, a specification on the type of damaged objects can be added. The possible values are:

- defence structure;
- road;
- bridge;
- hiking path;



(a) First part of the screen.

(b) Second part of the screen.

(c) Third part of the screen.

Figure 4.8: Info screen.

- house;
- building;
- agricultural building;
- movable property;
- cultural heritage;
- utility pole;
- other (with a specification).

Notes An optional open text field that allows the user to add any additional note.

4.2.4 Landslide manipulation

Once a landslide is inserted, independently from the application mode, a marker will appear on the map showing its position.

A click on a marker of this kind opens the information screen, where the user can visualize all the characteristics of the entry (figure 4.8). From here the application also gives the possibility to delete or edit the landslide.

4.2.5 Off-line mapping and synchronization

As already stated, the possibility of using the application without an Internet connection was one of the main concerns during the development of the phase. The final solution consists in leveraging the local database of the phone to temporarily save the data until the user synchronizes it with the remote database on the server.

Mapping

If the user tries to map a new landslide while she/he is off-line, the system will save the data locally. All the landslides saved locally will be displayed on the map using gray markers. Each modification of those entries (being it a deletion or an update) will only affect the local database, even if it is performed when the Internet connection is available.

If a user is currently signed in as a *guest* (see Section 4.2.1), she/he will only be able to visualize locally saved landslides and work with them.

Synchronization

At any time, given that an Internet connection is available and the user is properly logged in (i.e. not a guest), the application allows the user to synchronize the content of the local database with the server using the proper button (see Section 4.2.2).

This action will transfer all the landslides on the main database deleting them from the local one. Any sub-sequential modification applied to those data will be saved remotely.

4.3 Architectural design

LandslidesSurvey is designed as a *three-tier architecture*: a client-server architecture characterized by the logical and physical separation of the system in *presentation layer* (client), *business logic layer* (server), and *data layer* (database).

By modularizing the application into layers, this approach guaranties great flexibility, since each of the parts can be independently modified without the need to rework the entire system.

On top of that, the physical separation enables the developer to upgrade or replace any of the tiers to follow the changes in requirements or technology. For example, a change of the operating system in the presentation layer would only affect the user interface code.

Figure 4.9 shows a graphical representation of the architecture of LandslidesSurvey system. Further details about each of the layers are given in the following sections.

4.4 Client

The client corresponds to the *presentation layer* of the system architecture.

Since the majority of calculation and business logic are handled on server side, the client of LandslidesSurvey can be considered as a *thin client*. "Thin clients are network computers, NetPC's, or Windows-based terminals that rely on servers for application and processing power" [Sinclair and Merkow, 1999].

In other words, the client has only to display the information retrieved from the server through a dynamic GUI (Graphical User Interface).

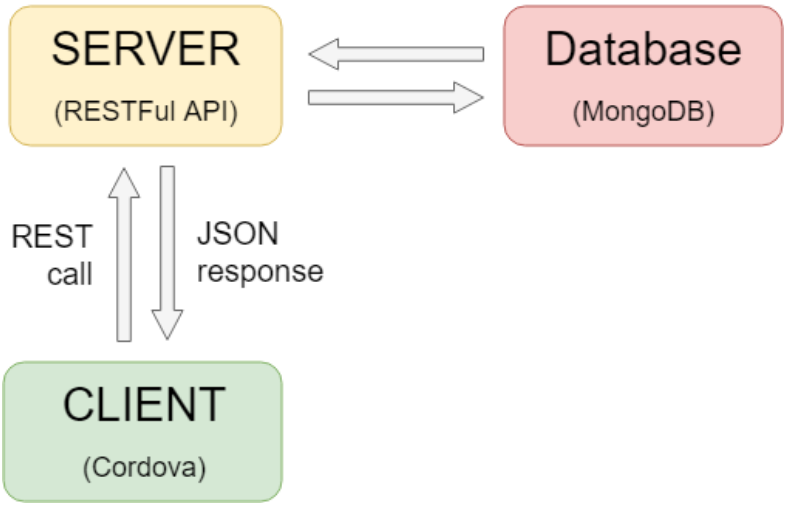


Figure 4.9: High-level view of LandslidesSurvey architecture.

4.4.1 Client-Server communication

The client communicates with the server through a series of *REST Calls*.

REST stands for *Representational State Transfer* and can be defined as "an abstraction of the architectural elements within a distributed hypermedia system" [R. T. Fielding and Taylor, 2000]. In practice, REST is an architectural style used to create Web services through a set of constraints that provide interoperability among computer systems on the Internet. The guiding principles of REST are:

- **Client-server architecture**: a separation of concerns to improve portability and scalability.
- **Statelessness**: the server never stores the client context. That means that each request from client to server must contain every information necessary to understand the request.
- **Cacheability**: clients and intermediaries can cache responses.
- **Layered system**: a client cannot tell if it is connected directly to the end server or to an intermediary along the way. This constrain allows an architecture to be composed of multiple hierarchical layers.

The server responds to these calls with the requested information in *JSON* format. JSON (JavaScript Object Notation) is a lightweight, easily readable data-interchange format. Although the format is language-interdependent, it is based on a subset of the JavaScript Programming Language and this makes particularly easy to parse it into JavaScript objects.

4.4.2 Coding details

In LandslidesSurvey the client is represented by the mobile application. One of the main objectives was to create a *cross-platform* system that could be used both on

Android and iOS phone to include as many users as possible in the data-gathering process.

Instead of relying on platform-specific languages - such as Java for Android and Swift for iOS - that present the key limitation of not being transferable to other systems, the application is built using *Apache Cordova*. This open-source mobile application development framework wraps a code written using HTML5, CSS[3] and JavaScript to create a system that can be run as a native application. One of the main advantages of Cordova is the possibility to leverage standards-compliant APIs that allow the developer to access the device capabilities such as sensors, data, and network status.

Compatibility

The way in which Cordova renders the user interface of its applications consists in using a "browser-based WebView [implemented] within the native mobile platform" [Apache Software Foundation, 2019].

This behaviour, however, poses some compatibility issues with JavaScript. Generally speaking, there exist different JavaScript versions identified by the versions of *ECMAScript* (ES), a scripting-language specification created to standardize JavaScript. Each new version of ECMAScript introduces new features to the language, but browsers (and consequently WebViews) usually do not support those additions right away.

It can be safely asserted that right now almost every phone is equipped with a WebView that supports ES5 functionalities, but the code of LandslidesSurvey relies on features introduced in next releases, such as classes (ES6) or async functions (ES8). To solve this compatibility problem, the application leverages on the open source tool *Babel*, a JavaScript compiler that is mainly used to convert ECMAScript 2015+ (ES6+) code into a backwards compatible version for older browsers or environments.

Through this tool it is also possible to combine all the code in a single, minified[4] file, reducing the size of it and, consequently, the loading time of the application.

Object-oriented programming and the Singleton pattern

The JavaScript code of LandslidesSurvey is written using a programming paradigm known as Object-Oriented Programming (OOP), whereby the program is modelled as a collection of collaborating objects. An *object* is an entity that possesses a state in the form of *attributes* and a behaviour defined by its *methods*. Those objects are instances of *classes* which can be seen as templates that define the objects structure and type.

On the other hand, the application is composed of a collection of *activities*, defined as "single, focused things that the user can do" [Google LLC, 2019]. For example, there will be a `MapActivity` containing the map itself and all the methods that handle the user interaction with it, or an `InfoActivity` used to

[3]For LandslidesSurvey a super-set of CSS called SCSS has been used.

[4]Minification refers to the process of removing unnecessary or redundant data without affecting how the resource is processed by the browser.

show the information about a single landslide. All the activities are modelled as classes, but it is imperative that only one instance of those classes exists at each stage of the application life cycle.

In software engineering, a software design pattern[5] called *singleton pattern* is used to ensure this property. This technique consists in hiding the constructor of the class and defining a public static method that returns the sole instance of the class. In this way, the instantiation of the class is restricted and only one instance can exist at any time.

Listing 4.1 shows how the pattern is implemented in JavaScript.

Listing 4.1: JavaScript singleton implementation.

```
class Singleton {

   // Private property representing the sole instance of
      the class
   static _instance;

   // The constructor should never be called directly
   constructor(){ ... }

   // Returns the sole instance of the class
   static getInstance() {

      // If no instance exists, create one
      if (!Singleton._instance)
         Singleton._instance = new Singleton();

      return Singleton._instance;

   }

}


/* Proof of concept */

let first  = new Singleton();
let second = new Singleton();

console.log(first === second); // Output: true
```

[5] A general and reusable solution to a common problem.

Map and geolocalization

Being able to locate the user at any time and in a precise way is one of the most important features of LandslidesSurvey . In order to do so, the application leverages the JavaScript *geolocalization API*, that allows the system to obtain the precise location of the user through both the GPS and the Wi-Fi of the phone. The API is able to retrieve the current coordinates (latitude and longitude) of the device as well as the precision of those coordinates, the altitude, and its precision.

Another key aspect to enhance the user experience is the possibility for the user to visualize herself/himself and the landslides she/he has inserted on a map. This is achieved through the usage of *Leaflet*, an open-source JavaScript library that allows the creation of interactive maps in an easy way.

Internationalization

In order to make the application accessible to the largest possible audience, particular attention was paid to the internationalization process, trying to ensure a the system that can "work well for, or (...) be easily adapted for, users from any culture, region, or language" [World Wide Web Consortium, 2019].

To achieve this goal, LandslidesSurvey leverages on the open-source framework *i18next*. i18next offers a flexible and scalable solution allowing the developer to prepare several JSON files containing the translations and to load them when needed. Advanced functions like interpolation, formatting and plurals are available, and a large collection of plug-ins make i18next suitable for a variety of different scenarios.

In particular, the application makes use of the *i18nextXHRBackend* plug-in to load the resources asynchronously and of the *jqueryI18next* plug-in to translate the DOM elements directly inside the HTML tags.

4.5 Server

The server plays the role of the *business logic layer* in LandslidesSurvey architecture. It is focused on the work flows, the business components, and the methods responsible for the communication with both the database and the client.

This layer of LandslidesSurvey is also written in JavaScript using *Node.js*, an open-source JavaScript run-time environment[6] that enables server-side scripting. On top of that, the system leverages on the framework *Express.js*, designed specifically to build Web applications and APIs.

4.5.1 RESTFul API

LandslidesSurvey server is written to be a *RESTFul API*. Generally speaking, a Web API is "the face of a Web service, directly listening and responding to client requests" [Masse, 2011]. In practice, a Web API exposes a series of publicly accessible *endpoints* that specify where resources lie and can be accessed. In this

[6] An engine that enables a software program to be executed on a computer system.

way a *request-response* message system is created with a client that accesses the endpoints through URIs[7] posting a HTTP request and a server that responds to this request typically using the JSON format.

The REST architecture underlined in Section 4.4.1 is commonly applied to the design of Web API, creating a so-called RESTFul API. One of the main characteristics of this kind of systems is to take advantage explicitly of HTTP methods to access resources via URL, as defined by the *RFC 2616 protocol*, a "document [that] specifies an Internet standards track protocol for the Internet community" [R. Fielding et al., 2019]. The main HTTP methods (also known as *verbs*) that are used in the application are detailed below.

- **GET**. This method is used to simply retrieve an information without any other effect. It is considered to be a *safe method*, since it does not change the state of the server.
- **POST**. This verb requests that the server accepts the entity enclosed in the request as a new subordinate of the resource identified by the URI. In other words, it is used to create a new resource, for example to be saved into the database.
- **PUT**. It works similarly to the POST method, but it is used to modify an already existing resource.
- **DELETE**. This method deletes a certain resource.

Listing 4.2 presents all the endpoints of LandslidesSurvey API.

Listing 4.2: API endpoints list.

```
/* Authorization endpoints */

// Inserts a new user into the database.
PUT /auth/signup

// Confirms the email of a user.
GET /auth/confirmation/:token

// Resends to a user a confirmation email.
POST /auth/confirmation/resend

// Logs in a user.
POST /auth/login

// Trigger the password reset procedure.
POST /auth/reset-password

// Renders the password reset page.
```

[7] Uniform Resource Identifiers.

```
GET /auth/new-password:token

// Resets the password of a user.
POST /auth/new-password


/* Landslides manipulation endpoints */

// Retrieves all the landslides from the database.
GET /landslide/get-all

// Retrieves all the landslides mapped by a user.
GET /landslide/user:userId

// Retrieves the information about a single landslide.
GET /landslide/:landslideId

// Inserts a new landslide into the database.
POST /landslide/post

// Update a landslide already saved into the database.
PUT /landslide/:landslideId

// Deletes a landslide form the database.
DELETE /landslide/:landslideId


/* User profile manipulation endpoints */

// Retrieves the information about a single user.
GET /profile/:userId

// Changes the email of a user.
PUT /profile/:userId/change-email

// Changes the password of a user.
PUT /profile/:userId/change-password

// Changes the information about a user.
PUT /profile/:userId/update-profile
```

4.5.2 Authentication flow

Since the application deals with personal data, one of the main concerns during its design and development was to guarantee an adequate level of security. To achieve

this goal, LandslidesSurvey makes use of the *OAuth 2.0* protocol to secure the registration process as well as the communication between client and server.

OAuth 2.0 is "the industry-standard protocol for authorization [that] focuses on client developer simplicity while providing specific authorization flows for Web applications, desktop applications, mobile phones, and living room devices" [IETF OAuth Working Group, 2019].

The sequence diagram in figure 4.10 shows the OAuth 2.0 flow implemented by LandslidesSurvey in a scenario where the user is not authenticated and wants to access a resource.The steps are detailed below:

1. The User opens the application; since she/he is not authenticated, she/he requests the login activity.
2. The client creates the activity and displays it to the User.
3. The User inserts her/his credentials (email and password).
4. The client sends the credentials to the server.
5. The server validates the credentials internally.
6. Having found that the credentials are correct, the server creates an Access Token encrypted with her/his private key. The Access Token has an expire date set to 24 hours, meaning that a session stays active for that period of time before the user has to login again.
7. The server sends back a response that contains the Access Token.
8. The client stores the Access Token into her/his local storage.
9. The client redirects the user to the application.
10. The User requests a generic activity.
11. The client retrieves the Access Token.
12. The client requests to the server the data needed for the activity and sends the Access Token as a part of the request.
13. The server validates the Access Token decrypting it with its private key.
14. Having found that the Access Token is valid, the server sends back the requested data.
15. The client populates the activity.
16. The client renders the activity.

4.6 Database

In the architecture of the system the database corresponds to the *data* layer. Its purpose is to store all the data coming from the application, allowing the system to perform the operations and to manage the functionalities related to persistent data.

LandslidesSurvey data storage is based on *MongoDB*, a widely used, dynamic and scalable *non-relational* database. Being NoSQL and object-oriented, MongoDB

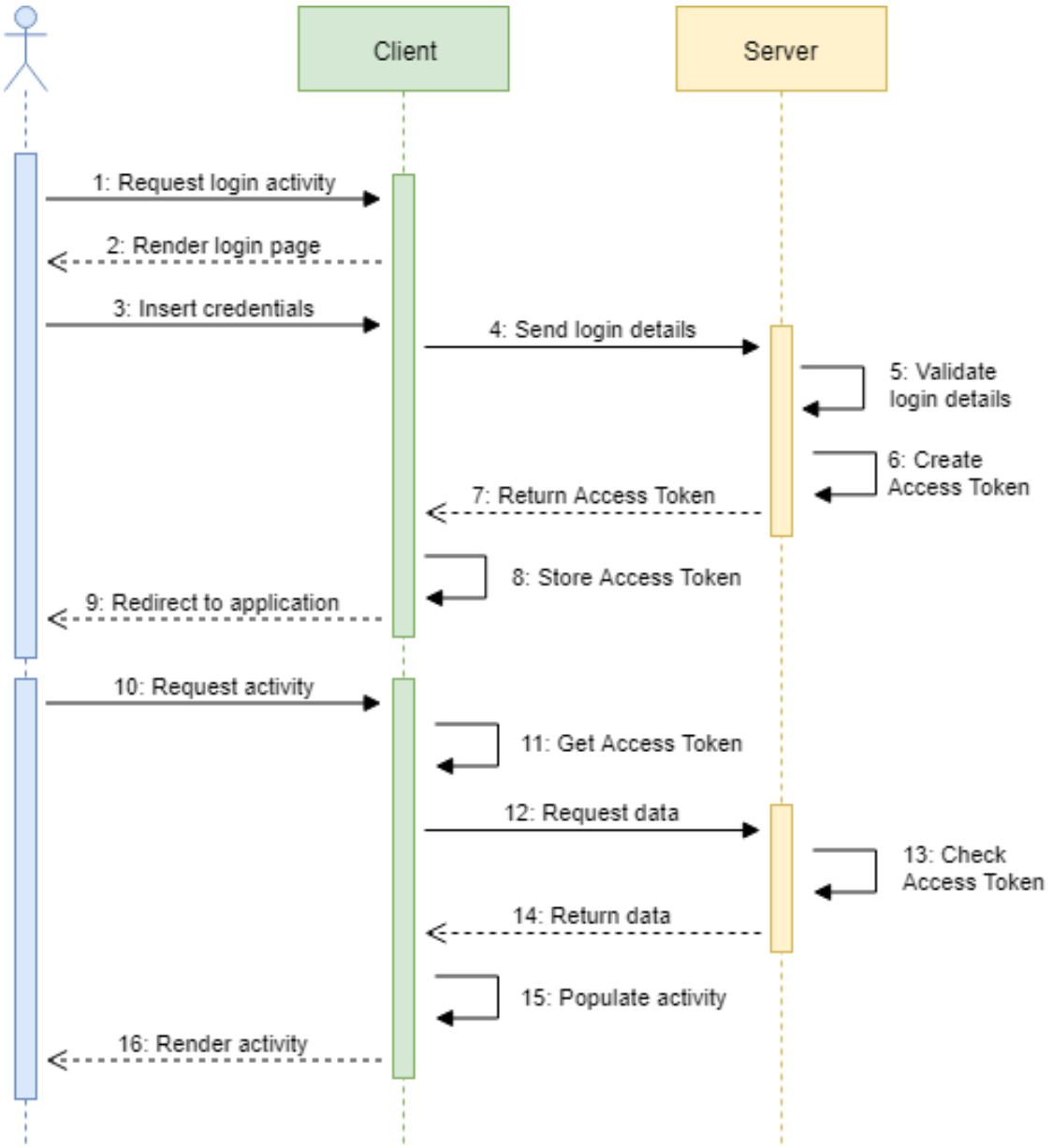


Figure 4.10: Sequence diagram of the OAuth 2.0 authentication flow.

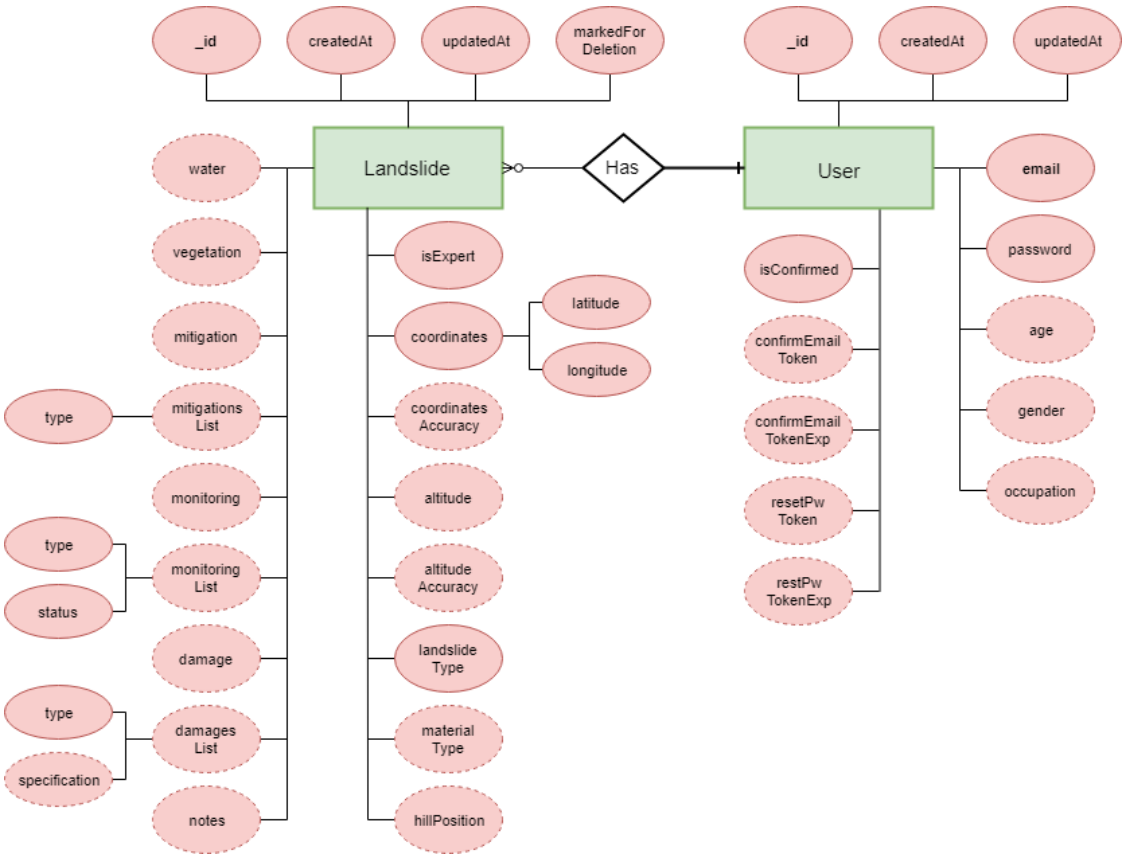


Figure 4.11: Entity-relationship diagram of the database.

stores data as documents in a binary representation called BSON (Binary JSON), instead of using tables like any traditional relational database.

This approach guaranties the flexibility that LandslidesSurvey needs, since on one hand the database schema presents a level of nesting (e.g. the lists of mitigations, monitoring systems or damaged objects) and on the other the volume of stored data is expected to grow considerably and non-relational databases present a simpler "horizontal" scalability with respect to relational ones.

These advantages usually come at the cost of sacrificing some of the critical capabilities of SQL database. However, one of the main strengths of MongoDB is that it is built to be "the only database that harnesses the innovations of NoSQL while maintaining the foundation of relational databases" [MongoDB, Inc., 2019]. MongoDB supports ACID transactions, implements ad hoc queries using a unique query language, and improves the performance of searches through a highly-functional secondary indexes system (including text search and geospatial indexing), making it the optimal choice for LandslidesSurvey data storage.

Figure 4.11 shows the high level entity-relationship diagram of the database.

4.6.1 Static objects storage

A different approach has to be taken for storing images. Since the application requires the users to upload a photo of each landslide they have mapped, storing

those objects into the database would largely increase its size, diminishing its performance and posing scalability issues.

The solution is to store all these large, static files into a separate *object storage*. This strategy subdivides the data into distinct units, or objects, and the content of the file becomes totally irrelevant to how it is stored. Moreover, objects are not placed in a hierarchy of directories, but they live in a flat address space, and the application can identify them by their unique address, saved in the database.

4.7 Code documentation

When writing a software, it is always of primary importance to produce a reliable code documentation, especially if it is an open API that could potentially be used by other developers for their projects. A well written documentation improves the quality of the product, cuts support costs, and eases the maintenance process.

For these reasons, the whole code of LandslidesSurvey is properly documented in a thorough but not verbose manner.

4.7.1 Client documentation

The client-side code of the system is documented through *JSDoc*, a markup language used to annotate JavaScript source code files. JSDoc syntax and semantics make it suitable to handle JavaScript dynamic behaviour. Listing 4.3 shows the documentation of a generic function as an example.

From the comments embedded in the source code, a tool called *JSDoc 3* was used to produce an HTML website from which the complete documentation can be consulted.

Listing 4.3: JSDoc documentation of a generic JavaScript function.

```
/**
 * Returns the sum of two numbers.
 *
 * @param {number} a - The first addend.
 * @param {number} b - The second addend.
 * @return {number} The sum of the two addends.
 */
function sum(a, b) {

    return a + b;

}
```

4.7.2 Server documentation

The documentation of the API is written following the *OpenAPI Specification* (OAS). OAS "defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic" [Miller et al., 2019].

The tool of choice to write the documentation is *ReDoc*. This free and open-source tool allows developers to deliver interactive, responsive and up-to-date API documentation deployable on any web page.

Chapter 5

Application testing

One of the main steps in developing a successful software is the *testing process*. A careful and extensive testing guarantees a software that behaves predictably and consistently, offering no surprises to its users. The quality of the product greatly increases through testing, and the maintenance process becomes easier and cheaper.

There are a lot of misconceptions about the intrinsic meaning of software testing. A test is not supposed to show that there are no errors or that the program is running correctly or as expected. This is not only impossible, since it would require to test an infinite set of cases, but also conceptually wrong. A test is supposed to add value to the software in terms of quality and reliability; this can be done only by finding and fixing errors.

Therefore, a correct definition could state that software testing is "the process of executing a program with the intent of finding errors" [Myers et al., 2011]. This means that a test is successful when it actually finds errors, allowing the developer to correct them and raising the reliability of the software.

With these concepts in mind, a testing plan for LandslidesSurvey was carefully developed both for the client and for the server side. The next sections highlight the planning and the execution of those tests through a variety of tools and techniques.

Section 5.1 presents the general implementation plan of the software. In Section 5.2 the actual testing plan is presented with an overview of the various strategies. Finally, Sections from 5.3 to 5.5 outline in detail the various phases of LandslidesSurvey testing process.

5.1 Implementation plan

Before starting with the actual testing, it is useful to make a small digression in order to analyse the *implementation plan* of LandslidesSurvey.

Figure 5.1 shows the main stages of the system implementation. As it can be seen, the development life cycle of LandslidesSurvey is based on an *iterative model*, an implementation strategy that leverages on *iterative design* and *incremental development*. The iterative model focuses on an initial, simple build which can be tested and evaluated. This simplified implementation is then progressively enhanced, making it more complex and increasing its feature set through a series of incremental alterations.

The main advantage of this approach lays in its strong adaptability. The frequent iterations make the model easily adaptable to any change in the requirements or in the code structure, diminishing the costs both in terms of money and of time.

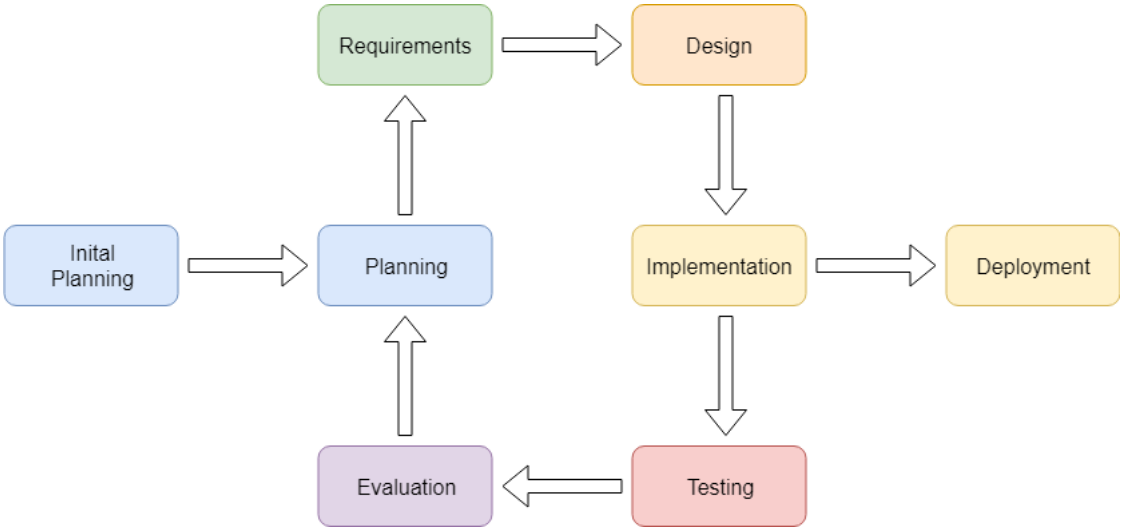


Figure 5.1: Schema of the iterative model.

5.1.1 Phases

Contrary to traditional approaches like the *waterfall model*, made of a rigid structure of sequential blocks, the iterative model is composed by a handful of stages repeated in a cycle that improves the software at each iteration.

Referring to figure 5.1, a detailed description of those stages follows.

- **Initial planning**. The starting phase during which the project begins to take shape. Here high level requirements are set and a general idea of the scope of the software is outlined.

- **Planning & requirements**. During this phase the specification documents are mapped out and the requirements for both software and hardware are set. In general, this stage lays the foundations for the whole iteration.

- **Design**. Once the requirements have been defined, a software solution that meets these requirements is designed. In this phase, either a new design is created or an existing one is extended.

- **Implementation**. Here the software is actually coded. All the plans, specifications and designs up to this point are implemented into the current iteration of the project.

- **Testing**. All the code written in the previous phase is now reviewed and tested. Any bug or issue that is found is fixed.

- **Evaluation**. Once all the other stages are completed, the software is evaluated by the stakeholders. This allows the developers and the clients to examine the status of the project and operate any change they see fit in the following iteration.

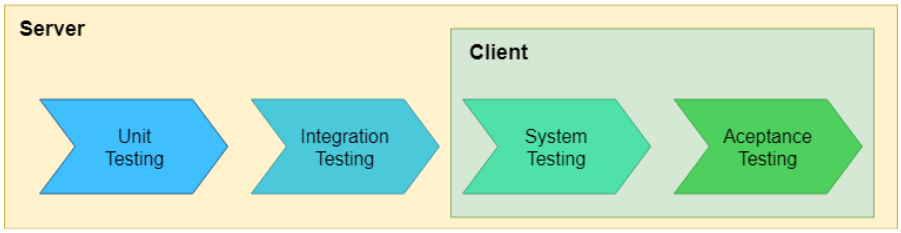


Figure 5.2: Testing plan of LandslidesSurvey.

5.2 Testing plan

The testing plan of LandslidesSurvey is built around four different levels as shown in figure 5.2: unit testing, integration testing, system testing and acceptance testing.

As the figure underlines, the first two levels are used to test only the server, while the last two are applied to the whole system. This is due to the fact that testing user interfaces with unit and integration testing is fairly difficult and often gives poor results. Therefore, it is better to leave those levels to the logic layer and let the users themselves test the front-end part of the system.

5.2.1 Testing strategies

An important specification has to be done regarding the testing strategies applied to each level. Two of the most prevalent strategies are traditionally the *black-box testing* and the *white-box testing* that differ from the point of view of the tester during the design of the test cases.

White-box testing

This testing strategy, also known as *structural testing*, aims to test the internal structure of the software. White-box testing operates directly at source code level, trying to cover the highest possible percentage of the code to discover any logical or structural error.

An internal perspective of the system is needed to design the test cases. Practically, the tester (usually the developer herself/himself) has to execute the various pieces of the system with controlled inputs to see if the outputs are the expected ones.

In the testing process of LandslidesSurvey, white-box testing was applied to the unit and integration testing levels.

Black-box testing

Black-box testing, or *data-driven testing*, does not bother with the internal behaviour of the program. Instead, it tries to find scenarios in which the system does not behave according to its specifications.

Test cases are built using specifications and requirements, and the tester is aware of *what* the software should do and not of *how* it effectively does it.

The system and the acceptance testing levels of LandslidesSurvey testing process are based on black-box testing.

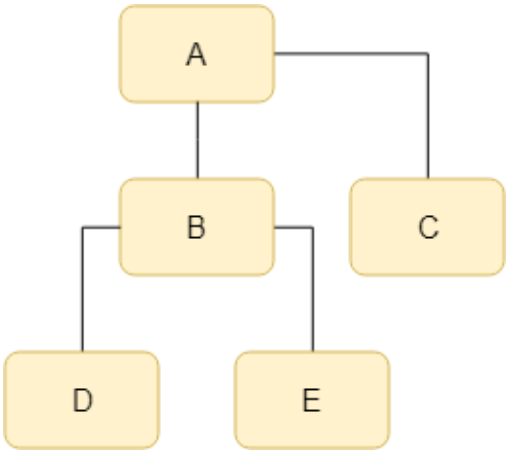


Figure 5.3: A simple module system.

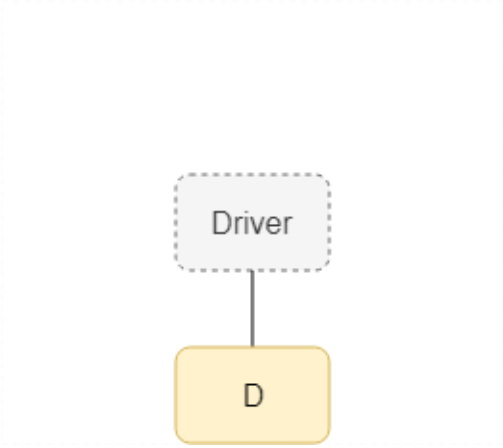


Figure 5.4: Intermediate state in the bottom-up test.

5.3 Unit and integration testing

The first level of testing, *unit testing*, is the process of testing the individual components, sub-programs and procedures in the system. On the other hand, *integration testing* consists in the combination among those individual components to test interfaces and modules interaction.

There are two approaches to integration testing:

- **"big-bang" integration** where tests begin only after the integration has been completed;
- **incremental integration** where tests occur while the component are being integrated.

Non-incremental testing requires less work at the cost of a poor observability, diagnosability, efficacy and feedback. For this reason, LandslidesSurvey is tested using an incremental approach leveraging a technique called *bottom-up testing*.

The bottom-up strategy consists in conducting first unit tests for the terminal modules of the program (i.e. modules that do not call other modules) and gradually moving upwards, integrating the components one by one. This integration process continues until all the modules are integrated and the entire application can be tested as a single component.

Figure 5.3 shows a simple hierarchical system composed by four modules. In a case like this, the first step would be to test modules C, D and E either serially or in parallel. Afterwards, modules D and E can be integrated and module B can be tested. The next step would be to integrate B and C and to test A and, finally, to test the whole system.

It is important to notice that in bottom-up integration each intermediate module needs a special *driver module* as shown in figure 5.4. This driver module is responsible for providing test inputs, calling the module that is tested, and displaying the outputs. Higher modules replace drivers and need their own drivers all the way up to the full integration.

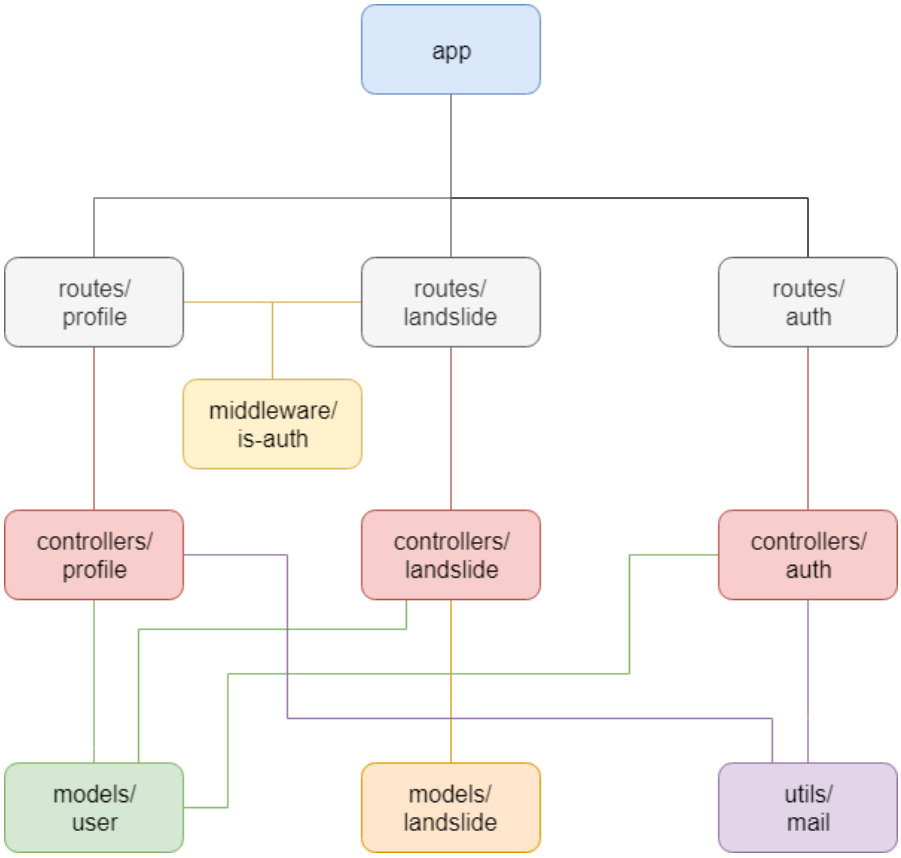


Figure 5.5: Module structure of LandslidesSurvey.

5.3.1 Implementation

The module structure of LandslidesSurvey is shown in figure 5.5. Starting from the bottom we have:

- **models/***. The models of *Mongoose*, the helper library used to abstract MongoDB interaction in Node.js projects. Those models define the structure of the collection schemas of the database.[1]
- **utils/mail**. Module that exports some utility function in order to deal with email creation and sending.
- **controllers/***. Collections of functions that process requests. They are in charge of dealing with the logic behind the requests, retrieving and manipulating data from the models and returning the appropriate responses to the caller.
- **middleware/is-auth**. This module is a *middleware*, a function executed as part of the application request-response cycle that calls the next middleware in the stack once its execution is completed. In this case, `is-auth` is invoked before the controllers and is in charge of checking if a request contains a valid session token.

[1] A collection in MongoDB is the equivalent of a table in a SQL database.

- **routes/***. The actual endpoints of the application. They handle the requests and invoke the right middlewares and controller functions.
- **app**. The main module responsible for starting the server and the connection with the database and for initializing the request-response cycle with all the routes and the correct middlewares.

In order to execute the unit and integration tests, LandslidesSurvey leverages on two widely-used, open-source libraries: *Mocha* and *Chai*. The first allows the developer to actually write the tests, acting as a driver, while the second provides interfaces for *assertions*. Tests are built by executing a function with controlled input and asserting which should be the expected output. If the real output meets the expectations, the test is passed; otherwise, it is failed.

Other key elements in unit and integration testing are *stubs*. A stub is a piece of code that substitutes other programming functionalities. In this context it is used to simulate the execution of third-party libraries or modules that should not be tested since they do not belong to the code actually written by the developer. The open-source library *Sinon* serves this purpose in LandslidesSurvey .

On top of that, the open-source library *Istanbul* is used to create a report about the code covered by the tests.

Listing 5.1 shows the tests written for the middleware `is-auth`. All tests are successful, and the coverage for the code is of 100%.

Listing 5.1: `is-auth` middleware tests.

```
describe("is-auth middleware", function () {

    it("should throw a 401 error if no authorization
       header is present", function () {

        // Custom request without an authorization header
        const req = { get: function (headerName) { return
           null } };

        // Expectations
        expect(isAuth.bind(this, req, {}, () => {}))
            .to.throw("Not authenticated.")
            .with.property("statusCode", 401);

    });

    it("should throw a 500 error if the authorization
       header is only one string", function () {

        // Custom request with an authorization header
           composed by only one string
```

```
        const req = { get: function (headerName) { return
            "test" } };

        // Expectations
        expect(isAuth.bind(this, req, {}, () => {}))
            .to.throw()
            .with.property("statusCode", 500);

    });

    it("should throw a 500 error if the token cannot be
        verified", function () {

        // Custom request with a wrong token
        const req = { get: function (headerName) { return
            "Bearer wrong-token" } };

        // Expectations
        expect(isAuth.bind(this, req, {}, () => {}))
            .to.throw()
            .with.property("statusCode", 500);

    });

    it("should throw a 401 error if the token is not
        valid", function () {

        // Custom request with an invalid token
        const req = { get: function (headerName) { return
            "Bearer invalid-token" } };

        // Create a stub for the verify method of jwt
        sinon.stub(jwt, "verify");

        // Make the stub return undefined
        jwt.verify.returns(undefined);

        // Expectations
        expect(isAuth.bind(this, req, {}, () => {}))
            .to.throw()
            .with.property("statusCode", 401);

        // Restore the normal behaviour of the method
        jwt.verify.restore();

    });
```

```
    it("should yield a userId after decoding the token",
       function () {

        // Custom request with an valid token
        const req = { get: function (headerName) { return
           "Bearer valid-token" } };

        // Create a stub for the verify method of jwt
        sinon.stub(jwt, "verify");

        // Make the stub return a valid user id
        jwt.verify.returns({ userId: "id" });

        // Call the middleware
        isAuth(req, {}, () => {});

        // Expectations
        expect(req).to.have.property("userId");
        expect(req).to.have.property("userId", "id");
        expect(jwt.verify.called).to.be.true;

        // Restore the normal behaviour of the method
        jwt.verify.restore();

    });

});
```

5.4 System testing

System testing is the process of testing the complete integrated software with the purpose to "compare the system or program to its original objectives" [Myers et al., 2011]. It consists of a series of black-box tests conduct to detect any inconsistency between the product and its functional and non-functional requirements.

5.4.1 System deployment

Starting from this level, the entirety of LandslidesSurvey begins to be tested. Since system testing should be carried out in an environment as close as possible to the production one, the different components of LandslidesSurvey were deployed using a variety of free tools that allow us to simulate the final architecture of the system.

Client deployment

At the time of writing, the application is available only for Android phones. In fact, even if the code is fully compatible with iOS devices, Apple requires some additional licences and checks before a working iOS application can be built.

Therefore, the testing was carried out on Android devices leveraging on the *Android Emulator* to assure a wide variety of models and OS versions. This tool can simulate Android devices on a computer providing almost all the capabilities of a real phone (e.g. location, network, incoming calls, etc.).

Server deployment

With regards to the API, a widely-used platform called *Heroku* was used to house the code. Heroku is a cloud *platform as a service*, a cloud computing model in which a third-party actor delivers the hardware and software needed to deploy an application over the internet.

Heroku supports a variety of programming languages and frameworks, including Node.js, and provides the developer with useful tools like logging and environmental variables support.

Database deployment

MongoDB offers developers a cloud solution called *MongoDB Atlas* to deploy a database. MongoDB instances deployed in this way are called *clusters*.

One of the main advantages of this structure is the fact that clusters can be configured to implement replication, providing redundancy and high data availability.

5.4.2 Test cases

As already stated, the system testing phase concerns the whole system composed by the application fully connected with the API and the database. In practice, the full-stack application is tested against four categories of test cases built with the purpose of finding any discrepancy between the program and its objectives.

Functionality testing

The most basic type of testing is the one meant to determine if the software implements correctly all the functions specified in the requirements.

Different use cases are developed trying to include as many facilities as possible and the user tries to perform them highlighting any flaw or failure.

Usability testing

This test category is focused on finding human-factor or usability problems. The system must be intuitive, easy to use and user-friendly; otherwise, it cannot be successful.

When testing for usability lots of aspects have to be taken into consideration like clearness of requests, consistency of the user interface, meaningfulness of responses, and usefulness of error messages.

Performance testing

The purpose of performance testing is to identify bottlenecks affecting response time, utilization and system throughput.

The software is tested under different workloads and configurations to find any possible scenario in which the program does not satisfy its performance objectives.

Compatibility testing

Since the application is supposed to be used on a variety of devices with different hardware and software characteristics, it is important to find any combination that can pose compatibility issues.

It was during this phase that the problem with Cordova WebViews and ECMAScript versions underlined in Section 4.4.2 was discovered and fixed.

5.5 Acceptance testing

The last level of testing is the *acceptance testing*, a "formal testing with respect to user needs [...] conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user [...] to determine whether or not to accept the system" [Van Veenendaal, 2010].

The acceptance testing is focused on the end user who acts as tester asked to verify if the solution works for her/him. This level of testing is often called *beta testing* and consists in the releasing of a version of the software that implements all the features but is likely to contain a certain number of known and unknown bugs. This *beta release* is delivered to a certain number of users who try the software in a real environment searching for issues and giving a direct feedback to the developer.

At the time of writing LandslidesSurvey is in beta phase. The system has undergone all the previous levels of testing and version 1.0.0 is available as a *pre-release*.

5.5.1 Client closed-beta test

Even if the system as a whole is in beta, the Client side of LandslidesSurvey has already passed this stage. During the month of May a standalone version of the application was distributed to a small group of students of the *Politecnico di Milano*.

The system was able to operate only locally, saving the landslides in the database of the phone, but the test revealed to be a great opportunity to validate both the functioning and the user-friendliness of the application.

The software was shipped to the students of the course of *Geologia Applicata 2* held by Professor Laura Longoni who accompanied them to the Tartano basin to collect some data about landslides. The test was a success: students were able to map twelve landslides in an easy and precise way. In the process some bugs and usability flaws were outlined.

5.5.2 Example of bug fixing

A real example of a bug, which was discovered during the aforementioned closed-beta distribution, is presented here together with its fixing in order to show the usefulness of beta testing.

The use of the application in a mountain area highlighted a problem related to the delay of the phone GPS system in locating the user precisely.

In fact, in order to save the battery, the application has been built to detach the position watcher when it is put on pause, just to reattach it as soon as it retrieves the focus. This happens for example when the user takes the picture of the landslide. Due to the poor signal, during the test the GPS was not able to retrieve the precise location in time when the user had taken the picture and inserted the data immediately afterwards. When it happened, the saved coordinates ended up being wrong.

Fixing the bug was fairly straightforward. Since then, the coordinates of the landslide are the ones recorded as soon as the *new landslide* button is pressed and not the ones recorded when the data are actually inserted.

Even if the fix process was easy, detecting this flow without a field test would have been very difficult. We can conclude that the beta test allowed us to ship a product with a higher usability and reliability.

Chapter 6

QGIS plugin

The data collected through LandslidesSurvey can be extremely useful for analytical studies and susceptibility assessment. For this reason, it is crucial to provide users with a way to easily explore and download all the database entries, which is not possible through the application alone.

During the development phase QGIS proved to be the most suitable platform for this purpose. The desktop GIS application, in fact, is widely used both by professionals and non-professionals thanks to its user friendliness, completeness and open-source nature.

Since the goal was to extend the native functionalities of the system, the ultimate decision was to create a plugin, a small software component that could fit inside the main application to add a specific feature. In order to maintain coherence with the rest of the developed software, the plugin was called LandslidesSurvey, too. The source code of the software can be found on GitHub.[1]

Section 6.1 presents in details the capabilities of the plugin, while in Section 6.2 the development process is outlined.

6.1 Features

The purpose of LandslidesSurvey QGIS plugin is to download and visualize all the landslides mapped using LandslidesSurvey application. The plugin allows the user to query the database retrieving up-to-date data, to spatially filter those data, and to download them both in JSON and in Shapefile format.

The plugin is available for any version of QGIS 3 and does not require any external module.

6.1.1 Outputs

The plugin connects directly to the open API detailed in Section 4.5.1 to retrieve all the data about the landslides from the database. Those data are returned in JSON format. However, given their nested structure, they are not suitable for being loaded in QGIS right away.

In QGIS, in fact, the standard vector format is the *Shapefile* developed by Esri. This open format stores the attribute data of its features in a `.dbf` file using the *dBase IV* format. This means that the attributes are stored in tables as flat structures, and multi-level elements cannot be represented like JSON does.

[1] https://github.com/epessina/LandslidesSurvey-QGIS-Plugin.

Listing 6.1 shows the structure of a landslide retrieved from the database. It would be easy enough to flatten simple arrays like the `coordinates` field that can be separated into two attributes: `latitude` and `longitude`. However, more complex constructions are impossible to be represented in a single table (e.g. the arrays of monitoring measures or damaged objects that are composed by a variable number of entries).

Listing 6.1: JSON structure of a landslide.

```
{
  "_id": "5d5ec397867dd2000423b682",
  "user": "5d5ac18efed53a00048e91c4",
  "markedForDeletion": false,
  "checked": false,
  "expert": true,
  "coordinates": [45.6017156, 8.9251083],
  "coordinatesAccuracy": 12.805999755859375,
  "altitude": -999,
  "altitudeAccuracy": 0,
  "type": "rotationalSlide",
  "materialType": "rock",
  "hillPosition": "atTheTop",
  "water": "dry",
  "vegetation": "grass",
  "mitigation": "yes",
    "mitigationList": [
    { "type": "rockfallBarriers" },
    { "type": "frictionNets" }
  ],
  "monitoring": "yes",
    "monitoringList": [
    { "type": "distometer", "status": "functioning" }
  ],
  "damages": "directDamage",
  "damagesList": [
    { "type": "building",  "specification": "" },
    { "type": "other", "specification": "Silos" }
  ],
  "notes": "",
  "imageUrl":
     "images/bf3c33a0-6bbd-4aae-8162-0c74aad12eb7",
  "createdAt": "2019-08-22T16:32:23.422Z",
  "updatedAt": "2019-08-22T16:32:23.422Z"
}
```

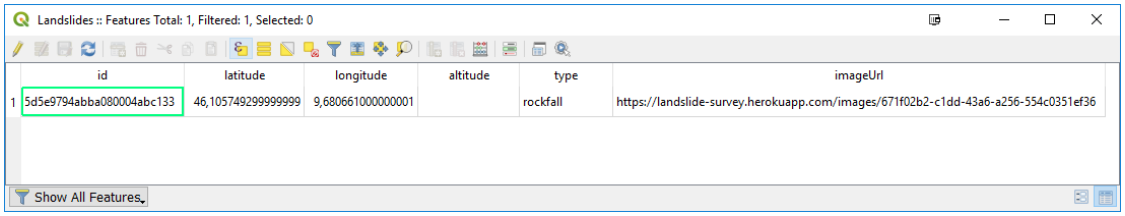


Figure 6.1: Attribute table of the Shapefile outputted by the plugin.

Figure 6.2: Tooltip displayed by the features of the Shapefile outputted by the plugin.

The plugin outputs two files to solve this problem:

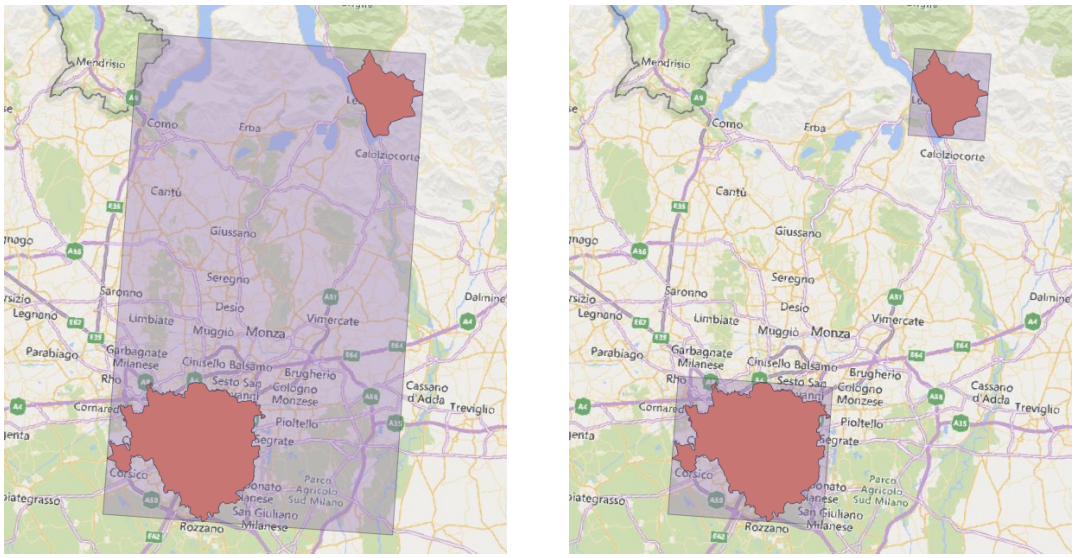
- a *JSON file* that simply contains all the data retrieved from the server;
- a *Shapefile* that displays all the landslides as points with a subset of the original attributes.

Figure 6.1 shows the *attribute table* of the output Shapefile.

Map tips

The Shapefile outputted by the plugin comes with a pre-defined *display property*. This property (accessible from the `Properties` menu of the layer) allows us to specify what is shown when the mouse is placed over a feature of the layer and the action *Show Map Tips* of QGIS is toggled on.

All the points created by LandslidesSurvey are configured to display the tooltip in figure 6.2. In this way the user can always have before her/him the information about the position of the landslide, the type of hazard, and the picture associated with the landslide.



(a) Single bounding box for all the features.

(b) Separate bounding box for each feature.

Figure 6.3: Difference between bounding box calculations.

6.1.2 Filtering

The amount of data saved in the database could be massive and possibly spread all over the globe. Therefore, it is likely that only a portion of those data is needed at any time, depending on the area of interest.

For this reason, LandslidesSurvey plugin gives the user the possibility to filter the results specifying a *bounding-box* and returning only the landslides that fall inside its boundaries.

The bounding box is calculated on the base of a vector layer chosen by the user. This layer can be already loaded in the project or can be selected on the fly from the file system. The bounding box calculation works with layers containing any kind of geometry (Points, Lines or Polygons), but Polygon geometries give the best and most consistent results.

This system is also designed to handle files with multiple geometries in a smart way. Instead of creating a single, big bounding box that covers all the features (figure 6.3a), the plugin computes a bounding box for each geometry (figure 6.3b) and tests the data against all the bounding boxes one by one. See Section 6.2.4 for more technical details.

6.1.3 User interface

The user interface of the plugin is rather simple, considering that most of the functionalities do not depend on user's inputs. Figure 6.4 shows the plugin dialogue. The various elements of the interface are detailed below.

1. **Bounding box check box**. Through this check box the user can decide whether she/he wants to filter the data or not. As long as the checkbox is unchecked, the user is not able to select a layer as bounding box and if the

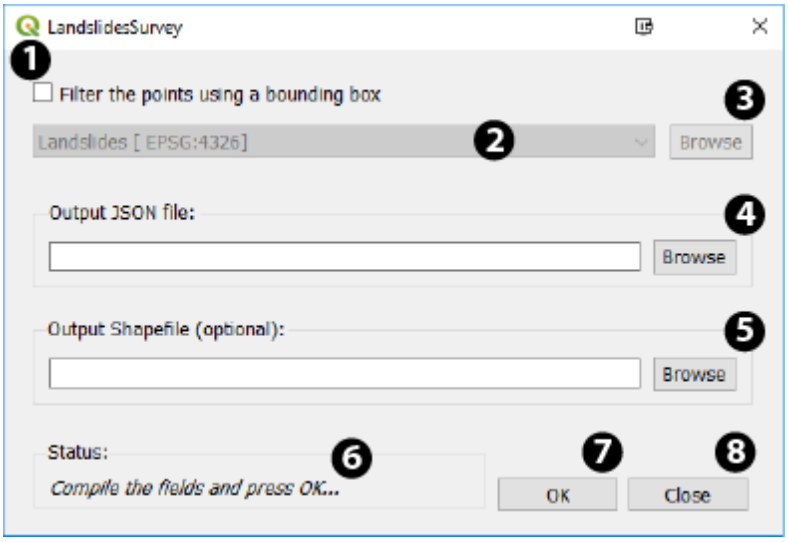


Figure 6.4: User interface of the plugin.

layer has already been selected, the system will not consider it during the computation.

2. **Bounding box combo box**. This drop-down menu contains all the vector layers currently in the project and allows the user to choose the one to use as bounding box.
3. **Bounding box tool button**. A click on this button opens a file system window from which the user can select a vector layer not currently in the project. The layer is loaded, displayed on the map and selected as bounding box.
4. **JSON output tool button**. From this button the user can open a file system window to set the path and the name of the JSON file that will be created by the plugin. This field is mandatory and if the user does not set it, the execution of the plugin will be prevented.
5. **Shapefile output tool button**. Just like the button for the JSON output, this widget allows the user to set path and name of the output Shapefile. The file is automatically loaded into the project when the plugin execution is completed. It is an optional field and leaving it empty will just tell the plugin not to create the file.
6. **Status label**. This read-only text field shows the advancement status of the process.
7. **Run button**. Button to start the computation.
8. **Close button**. Button to close the plugin.

6.2 Development

QGIS plugins are usually written in *Python*, and LandslidesSurvey is no exception. QGIS offers developers a blending between its functionalities and Python. Through

this blending, which is called *PyQGIS*, it is possible to create scripts, plugins, and standalone applications in order to implement new features or perform automated tasks.

LandslidesSurvey plugin is written for QGIS 3, the new version of the program based on Python 3.

6.2.1 System configuration

Generally speaking, a correct set-up of the development environment is key to writing and debugging code easily and efficiently. This is even truer for QGIS plugins, since the IDE[2] in use must have the same environment settings and use the same libraries and interpreter as QGIS.

PyCharm - a popular IDE distributed by JetBrains - was chosen to develop LandslidesSurvey plugin. Being Windows the underlying operating system, the easiest way to set-up the environment correctly was to create a *batch file*. A batch file is a script file composed by a series of commands that can be executed by the command-line interpreter of Windows. Listing 6.2 shows the content of the file `ls.bat`. Its main purpose is to set the paths for all the libraries needed by the plugin, making PyCharm aware of the functionalities offered by PyQGIS. At the end it runs the IDE.

Listing 6.2: IDE configuration Batch file.

```
:: Hide the comments
@echo off

SET OSGEO4W_ROOT=D:\OSGeo4W64
call "%OSGEO4W_ROOT%"\bin\o4w_env.bat

@echo off
path %PATH%;%OSGEO4W_ROOT%\apps\qgis\bin
path %PATH%;%OSGEO4W_ROOT%\apps\grass\grass-7.4.0\lib
path %PATH%;D:\OSGeo4W64\apps\Qt5\bin
path %PATH%;D:\OSGeo4W64\apps\Python37\Scripts

set
   PYTHONPATH=%PYTHONPATH%;%OSGEO4W_ROOT%\apps\qgis\python
set PYTHONHOME=%OSGEO4W_ROOT%\apps\Python37

set PATH=D:\Program Files\Git\bin;%PATH%

start "PyCharm aware of QGIS" /B "D:\Program
   Files\JetBrains\PyCharm 2019.2\bin\pycharm64.exe" %*
```

[2]Integrated Development Environment. An application that provides comprehensive facilities for software development.

6.2.2 Plugin Builder and file structure

Plugin Builder was used to ease the creation of LandslidesSurvey plugin. This tool, which is a QGIS plugin itself, generates a plugin template with all the files and directories needed to start the development process.

Listing 6.3 shows the complete directory structure of LandslidesSurvey plugins, while a detailed explanation of the most important files is presented below.

- `__init__.py`. The package initialization file and the starting point of the plugin. It must contain the method `classFactory(iface)`.
- `landslides_survey.py`. The main code of the plugin. It contains the methods for initializing the user interface and for handling the computations.
- `landslides_survey_dialog_base.ui`. The GUI of the plugin created by Qt Designer (see Section 6.2.3).
- `landslides_survey_dialog.py`. The compiled version of the plugin GUI.
- `metadata.txt`. The file containing the metadata of the plugin.
- `pb_tool.cfg`. The configuration file of *pb_ tool*, a command line tool to compile and deploy the plugins.
- `resources.qrc`. The `.xml` file created by Qt Designer. It contains the relative paths to the resources of the forms that compose the GUI.
- `resources.py`. The compiled version of the file described above.

Listing 6.3: Directory structure of LandslidesSurvey plugin.

```
landslides_survey/
   |-- help/
   |-- i18n/
   |-- scripts/
   |-- __init__.py
   |-- icon.png
   |-- landslides_survey.py
   |-- landslides_survey_dialog_base.ui
   |-- landslides_survey.py
   |-- Makefile
   |-- metadata.txt
   |-- pb_tool.cfg
   |-- plugin_upload.py
   |-- pylintrc
   |-- resources.qrc
   |-- resources.py
```

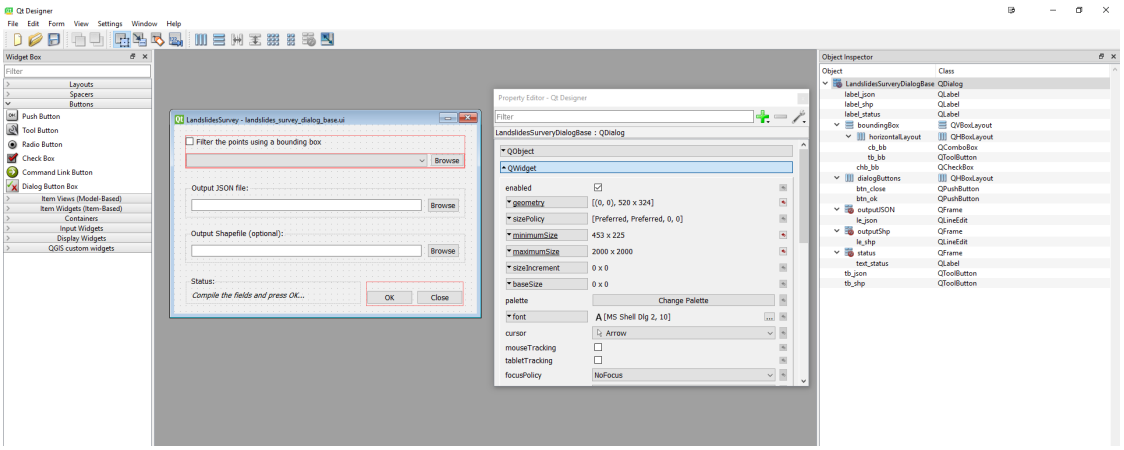


Figure 6.5: Qt designer.

6.2.3 User interface creation

The user interface of the plugin is created adopting *Qt Designer*, "the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets" [Qt Company Ltd., 2019].

Qt is a widely-used, free and open-source widget toolkit that makes available to developers a set of UI elements (called *widgets*) to create classic desktop-type interfaces. Qt designer - showcased in figure 6.5 - collects all those widgets in a simple and user-friendly UI builder that allows the design of dialogues on the fly, with the possibility of fully customizing their look through CSS style sheets.

Leveraging on Qt for QGIS plugins, however, raises a compatibility issue. Qt, in fact, is developed in C++ and needs *bindings* to be used for Python-based plugins. Language bindings are wrapper libraries that work as bridges between two programming languages. In particular, PyQGIS bindings in QGIS 3 depend on *PyQt5*, a comprehensive set of Python bindings for the fifth version of Qt.

Practically speaking, the `.ui` files created with Qt Designer are complied by PyQt5 in the correspondent Python version that can be used in the plugin code.

6.2.4 Bounding box calculation

As already stated, one of the main features of the plugin is the possibility of spatially filtering the landslides using a layer, handling multiple non-connected features in a correct way.

In practice, this task is accomplished in two phases.

1. The system retrieves all the features from the layer chosen by the user. Particular attention has to be put on the *CRS* of the layer. To correctly filter the data, the bounding boxes must be expressed in the same coordinate system of the landslides, which is *WGS84 (EPSG:4326)*, the last revision of the World Geodetic System. Therefore, the system performs the transformation if needed and saves the minimum and maximum coordinates of each bounding box inside an array. Listing 6.4 shows the function that handles this first step.

2. After having retrieved the landslides, the system loops for each entry through all the saved bounding boxes and keeps only the data whose coordinates fall inside one of them. This procedure is shown in listing 6.5.

Listing 6.4: Function to compute the bounding boxes.

```
def compute_bb(self):

    [...]

    # Initialize the array that will contain the bounding
       boxes
    boxes = []

    # Extract the CRS from the layer
    source_crs = self.bb_layer.crs()

    # Set the destination CRS as WGS84
    destination_crs = QgsCoordinateReferenceSystem(4326)

    # Initialize the transformation object
    transformation = None

    # If the CRS of the layer is not already WGS84
    if source_crs != destination_crs:

            # Compute the transformation
            transformation =
               QgsCoordinateTransform(source_crs,
               destination_crs, QgsProject.instance())

    # Extract all the features from the layer
    features = self.bb_layer.getFeatures()

    # For each feature
    for feature in features:

       # Calculate the bounding box of the feature and
          append it to the array
       boxes.append(feature.geometry().boundingBox())

    # For each bounding box
    for box in boxes:

       # If there is a transformation to be applied
```

```
        if transformation is not None:

            # Transform the CRS of the box
            box = transformation.transformBoundingBox(box)

        # Save the minimum and maximum coordinates of the
            bounding box
        self.bb_coord.append([box.xMinimum(),
            box.yMinimum(), box.xMaximum(), box.yMaximum()])
```

Listing 6.5: Function to filter the points.

```
def save_points(self):

    # Retrieve the data from the API
    res = requests.get(self.DB_URL + "landslide/get-all",
     headers = {"Accept": "application/json"}).json()

    # Initialize the array that will contain the filtered
        landslides
    landslides = []

    [...]

    # For each landslide
    for ls in res["landslides"]:

        [...]

        # For each bounding box
        for box in self.bb_coord:

            # If the coordinates of the landslide falls
                inside the bounding box
            if box[0] <= ls["coordinates"][1] <= box[2] and
                box[1] <= ls["coordinates"][0] <= box[3]:

                # Append the landslide to the array
                landslides.append(ls)

                    break

    [...]
```

Chapter 7

Conclusions

As stated in the introduction, shallow landslides represent one of the most frequent and widespread natural hazard and pose a large concern for ecosystem processes and anthropic activities.

For this reason, the ultimate goal of project MHYCONOS is the creation of a shallow landslides susceptibility map which can prove itself to be a rather effective and easily understandable tool for risk mitigation.

The work detailed in this thesis had the purpose to provide the project with an adequate IT support, crucial to present the results in a way which can be used to rise societal awareness on the causes and danger of shallow landslides.

At the beginning of the work the main concern was the creation of a database designed to house the data needed by the project partners. This task was accomplished through the free and open-source relational DBMS *PostgreSQL* enhanced with the *PostGIS* spatial extension. The result was a flexible spatial database containing all the information useful to explore the area of interest and now ready to store the future outcomes of the project.

The next step consisted in the development of a WebGIS capable of presenting those data in a clear and user-friendly way. The well-established, open-source server *GeoServer* was chosen as the core of the application. The client, on the other hand, was written using the open-source JavaScript library *OpenLayers*. This setup allowed the creation of a direct connection among database, server, and client. In other words, any data change performed by the project partners is now propagated seamlessly, transparently and immediately to the end user.

After having fulfilled the requests about data storage and visualization, the focus moved on citizen science and on the involvement of the general public into the data gathering process. For this purpose, the cross-platform mobile application LandslidesSurvey was developed and carefully tested. The implementation of the system followed a three-tier architecture comprised of the application as client, of a RESTFul API as server, and of a non-relational database.

The work ended with the creation of a plugin for *QGIS* built to allow the retrieval, filter and download of the data inserted through LandslidesSurvey mobile application.

7.1 Future developments

All the software developed for the thesis are fully functioning and fulfil the requirements set by project MHYCONOS. There is, however, room for improvement both in terms of implemented functionalities and of user experience.

WebGIS

One important feature missing in the WebGIS is the possibility to query the layers in order to obtain some information about the single features that compose the datasets. Together with this mechanic, future releases will also implement a download system for the various data.

Moreover, it could be useful to add an *info* section containing the history of the project and some tips on how to use and understand the WebGIS itself.

LandslidesSurvey application

The main improvement needed by the application is the creation of a system of tile caching to enhance the offline usability. Right now, without an internet connection available, the background map would not load, leaving the user with no clue about her/his current position. Even if the application works perfectly without a loaded map, the user experience would be greatly improved by the addition of the possibility to select an area and temporarily save the tiles.

Another possible development is related to the picture taking process. Future releases will let the user add up to three photos and will register the orientation and tilt of the phone during the process.

QGIS plugin

The plugin could certainly benefit from a better user interface with enhanced error communication and a *log* panel to keep track of the operations.

Bibliography

Apache Software Foundation

2019 *Cordova documentation, "Embedding WebViews"*, Aug. 2019, https://cordova.apache.org/docs/en/latest/guide/hybrid/webviews/index.html. (Cit. on p. 35.)

Black, MA and WE Cartwright

2005 "Web cartography & web-enabled geographic information systems (GIS). New possibilities, new challenges", in *Proceedings of the 22nd International Cartographic Conference*.

Bosnic, Stefan, Ištvan Papp, and Sebastian Novak

2016 "The development of hybrid mobile applications with Apache Cordova", in *2016 24th Telecommunications Forum (TELFOR)*, IEEE, pp. 1-4.

Brovelli, Maria Antonia and Diego Magni

2003 "An archaeological web GIS application based on Mapserver and PostGIS", *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34, 5/W12, pp. 89-94.

Caine, Nel

1980 "The rainfall intensity-duration control of shallow landslides and debris flows", *Geografiska annaler: series A, physical geography*, 62, 1-2, pp. 23-27.

Ceriani, M and D Fossati

2006 "Inventario dei Fenomeni Franosi in Lombardia Regione Lombardia", *DG Protezione civile, Programma delle Ricerche Strategiche*. (Cit. on p. 1.)

Connolly, Thomas M and Carolyn E Begg

2005 *Database systems: a practical approach to design, implementation, and management*, Pearson Education. (Cit. on p. 9.)

Environmental System Research Institute

2019 *ArcGIS Enterprise, About web GIS*, Sept. 2019, https://enterprise.arcgis.com/en/server/latest/create-web-apps/windows/about-web-gis.htm. (Cit. on p. 15.)

Fielding, R., U. C. Irvine, J. Gettys, et al.

2019 *Hypertext Transfer Protocol – HTTP/1.1*, Aug. 2019, https://tools.ietf.org/html/rfc2616. (Cit. on p. 38.)

Fielding, Roy T and Richard N Taylor

2000 *Architectural styles and the design of network-based software architectures*, University of California, Irvine Doctoral dissertation, vol. 7. (Cit. on p. 34.)

Fowler, Martin

2002 *Patterns of enterprise application architecture*, Addison-Wesley Longman Publishing Co., Inc.

Google LLC

2019 *Android documentation, "Activity"*, Aug. 2019, https://developer.android.com/reference/android/app/Activity. (Cit. on p. 35.)

IETF OAuth Working Group

2019 *OAuth 2.0 documentation*, Aug. 2019, https://oauth.net/2/. (Cit. on p. 40.)

Infrastructure for Spatial Information in Europe

2019 *INSPIRE registry, Coordinate reference systems*, Aug. 2019, http://inspire.ec.europa.eu/theme/rs/. (Cit. on p. 8.)

International Organization for Standardization

2015 *Information technology – Vocabulary*, tech. rep., International Organization for Standardization. (Cit. on p. 16.)

Irwin, Alan

2002 *Citizen science: A study of people, expertise and sustainable development*, Routledge. (Cit. on p. 23.)

Juang, Caroline S, Thomas A Stanley, and Dalia B Kirschbaum

2019 "Using citizen science to expand the global map of landslides: Introducing the Cooperative Open Online Landslide Repository (COOLR)", *PloS one*, 14, 7, e0218657. (Cit. on p. 4.)

Kraak, Menno Jan

2001 "Settings and needs for web cartography", in *Webcartography: developments and prospects*, Taylor & Francis, pp. 1-8.

Kreuzer, Thomas M, Martina Wilde, Birgit Terhorst, and Bodo Damm

2017 "A landslide inventory system as a base for automated process and risk analyses", *Earth Science Informatics*, 10, 4, pp. 507-515. (Cit. on p. 4.)

Larman, Craig

2004 *Agile and iterative development: a manager's guide*, Addison-Wesley Professional.

Masse, Mark

2011 *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*, O'Reilly Media, Inc. (Cit. on p. 37.)

Mikoš, Matjaž, Željko Arbanas, Yueping Yin, and Kyoji Sassa
2017 *Advancing Culture of Living with Landslides: Volume 3 Advances in Landslide Technology*, Springer. (Cit. on p. 4.)

Miller, D., J. Whitlock, M. Gardiner, et al.
2019 *OpenAPI Specification, Version 3.0.2*, Sept. 2019, http://spec.openapis.org/oas/v3.0.2. (Cit. on p. 44.)

MongoDB, Inc.
2019 *MongoDB documentation, "Relational Vs Non Relational Database"*, Aug. 2019, https://www.mongodb.com/scale/relational-vs-non-relational-database. (Cit. on p. 42.)

Myers, Glenford J, Corey Sandler, and Tom Badgett
2011 *The art of software testing*, John Wiley & Sons. (Cit. on pp. 45, 52.)

Olyazadeh, Roya, Karen Sudmeier-Rieux, Michel Jaboyedoff, Marc-Henri Derron, and Sanjaya Devkota
2017 "An offline–online Web-GIS Android application for fast data acquisition of landslide hazard and risk", *Natural Hazards and Earth System Sciences*, 17, 4, p. 549. (Cit. on p. 4.)

Qt Company Ltd.
2019 *Qt Designer Manual*, Aug. 2019, https://doc.qt.io/qt-5/qtdesigner-manual.html. (Cit. on p. 64.)

Rosser, Brenda, Sally Dellow, Soren Haubrock, and Phil Glassey
2017 "New Zealand's national landslide database", *Landslides*, 14, 6, pp. 1949-1959. (Cit. on p. 4.)

Sinclair, Joseph T and Mark Merkow
1999 *Thin Clients clearly explained*, Morgan Kaufmann Publishers Inc. (Cit. on p. 33.)

Van Veenendaal, Erik
2010 "Standard Glossary of Terms used in Software Testing", *International Software Testing Qualifications Board*. (Cit. on p. 54.)

Whiteside, Arliss and Jim Greenwood
2010 *OGC Web Service Common Implementation Specification*, tech. rep., Open Geospatial Consortium.

World Wide Web Consortium
2019 *W3C Standards, "Internationalization"*, Aug. 2019, https://www.w3.org/standards/webdesign/i18n. (Cit. on p. 37.)