Deliverable D6.3
GIS-based web application

Submission date : 17 June 2018
Lead contractor

UIC

Contributors

ADS

Project Coordinator

University of Sheffield, USFD
**Executive Summary**

The purpose of this deliverable is to present the GIS tool where the managed data mainly targets transport (infrastructure) planning.

This tool has been designed to meet the strategic (transport, maintenance) and planning needs of users and is not intended to provide the precision required for operational purposes. However, the data on the rail network is provided both at the macro level (stations and lines, with low accuracy) as well as at the micro level (tracks, with high accuracy).

The GIS-based web application (complete with specifications and documented source code) is intended to help and determine the transport and commercial revenue potentials in the context of:

1) the human and industrial geography,
2) general economic data,
3) line characteristics, station and terminal locations,
4) observed traffic levels.

Research activities carried out under Task 6.2 focused on the overall design of the GIS-based web application and the design of each component in order to create a dynamic application support for managed data mainly targets transport (infrastructure) planning.

The design activity was mainly focused on the following actions:

- Design of the logical processing components (business logic) of the GIS application;
- Design of the user interface for GIS application;
- Design of the database of the GIS application;
- Design of the logical processing components (business logic) of the GIS web interface.

Functional specifications, technical specifications, and system architecture are developed by two sub-activities:

- Design of the specifications involving definition of functional and technical requirements of the system in concrete terms (functions, processing, interfaces, etc.);
- Design of the system architecture that includes the reference model, consolidated data model, logical and physical architecture, and design of the system warehouse.

The ambition was in particular to be able to handle the most up-to-date infrastructure description (from the European Register of Infrastructure, public data made available by the EU Agency for Railways), together with first-rate passenger train data derived from MERITS (a timetable data exchange activity established under UIC, grouping about 40 rail passenger operators, and currently opening data access to third parties).

The NeTIRail GIS-based web is the core component for developing the Decision Support Tools, which will be developed under Task T6.3.

The work carried out within this deliverable matches the task description for task 6.2 – decision support toolset, and there are no technical deviations. The only deviation is with regard to the submission date of this deliverable.
# Table of contents

Executive Summary .................................................................................................................. 3  
Table of contents ...................................................................................................................... 4  
Abbreviations and acronyms ..................................................................................................... 5  
1 Introduction ............................................................................................................................... 6  
   1.1 Working Methodology ........................................................................................................ 6  
   1.2 Technical Approach ........................................................................................................... 6  
2 Requirements for GIS-based web application .......................................................................... 8  
   2.1 Requirements for Web GIS application .............................................................................. 8  
   2.2 Requirements for quality attributes of systems ................................................................. 9  
3 Purposes of system for GIS-based web application ................................................................. 10  
   3.1 Global architecture of system ........................................................................................... 10  
      3.1.1 Database architecture ................................................................................................. 12  
      3.1.2 NeTIRail GIS-based web application system architecture ........................................ 15  
         3.1.2.1 Business logic and access data layer ..................................................................... 17  
         3.1.2.2 Client-side application layer .................................................................................. 20  
   3.2 NeTIRail GIS-based web application ................................................................................ 22  
      3.2.1 Geographic layer .......................................................................................................... 22  
         3.2.1.1 Administrative Regions layer ................................................................................ 23  
         3.2.1.2 Railway Network layer .......................................................................................... 25  
         3.2.1.3 Rail Stations layer .................................................................................................. 27  
         3.2.1.4 Population layer ..................................................................................................... 28  
         3.2.1.5 Land Use Layer ..................................................................................................... 32  
         3.2.1.6 Places layer ............................................................................................................ 33  
         3.2.1.7 Land Use Layer ..................................................................................................... 34  
      3.2.2 NeTIRail GIS-based web functionality ........................................................................ 35  
4 Testing of GIS-based web functionality .................................................................................. 39  
   4.1 Unit Tests and Component Tests ....................................................................................... 39  
   4.2 Prototype test in real conditions ....................................................................................... 43  
5 Conclusions .............................................................................................................................. 46  
6 References ................................................................................................................................ 47
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation / Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS</td>
<td>ADS-Electronic Research SRL</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript And XML, is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>NUTS</td>
<td>Nomenclature of Territorial Units for Statistics</td>
</tr>
<tr>
<td>ReST</td>
<td>Representational state transfer - providing interoperability between computer systems on the Internet. Using HTTP, the kind of operations available include those predefined by the CRUD HTTP methods GET, POST, PUT, DELETE</td>
</tr>
<tr>
<td>RINF</td>
<td>The European Register of Infrastructure</td>
</tr>
<tr>
<td>RTM</td>
<td>RailTopoModel (RTM) is a logical object model to standardise the representation of railway infrastructure-related data.</td>
</tr>
<tr>
<td>OSM</td>
<td>Open Street Map</td>
</tr>
<tr>
<td>SRID</td>
<td>A spatial reference identifier (SRID) is a unique identifier associated with a specific coordinate system, tolerance, and resolution.</td>
</tr>
<tr>
<td>UIC</td>
<td>International Union of Railway</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>WGS</td>
<td>World Geodetic System</td>
</tr>
</tbody>
</table>
1 Introduction

This report represents the deliverable: “D6.3 GIS-based web application”.

The objectives of the deliverable are to describe the following:

- Functional specification that provide a general description of the system including the actors, system functions, data model consolidated and database structure;
- Reference architecture;
- Logical and physical architecture;
- Main user interfaces.

1.1 Working Methodology

For research activities performed within the Task 6.2, the research team used the following working methodology:

- Evaluation existing techniques and solutions:
  
The work started with the evaluation of technologies, tools, algorithms, solutions and techniques used in GIS desktop application (e.g. QGIS or ArcGIS in order to create our GIS layer) and respectively of our GIS Web application design.
- System design:
  
The design took into account the results of the performed assessment step and sought to improve system performance by the defined configuration solution and to consider the best algorithms for use in various operational scenarios. The design was based on the specification of technologies and techniques for each type of system functionality.
- Risk Assessment:
  
A high-level design of an alternative, if unexpected obstacles arise in the initial technical design (such as failure in implementation, inadequate results, etc.) was taken into account when they were selected to use technological solutions (e.g. the possibility uses or import data from external geographical data sources, see section 3.2.1) to achieve the system.

Important decision points of passage or not (Go, No-Go milestones) from one activity / stage to another will facilitate decision making and the possible switch to an alternative technique.

1.2 Technical Approach

- System lifecycle

Lifecycle (see Figure 1-1) of the system consists of the following steps:

1. Requirements - Gathering of the requirements
2. Specification - Define functional requirements of the system;
3. Design - System design;
4. Implementation - implementation of the system;
5. Integration - Integration of the components and verification of integrated system;
6. Maintenance - Maintenance of the system, removal of any abnormal functionalities and formulating of the development needs;
7. Update requirements - Update system requirements and launch a process of its development.

![System development lifecycle diagram]

**Figure 1-1 System development lifecycle**

- **Aspects considered:**

The design stage of the Web GIS application focuses on the following aspects:

- The first is to consider the requirements;
- Development of the functional specification which assume the delimitation of functional requirements in concrete terms namely, internal processing, interfaces that are able to meet all requirements;
- Design of the system architecture which will include a reference model, data model consolidated logical and physical architecture;
- Design and specification of the implementation in detail for each component, including objects and methods to be implemented and interfaces to be provided:
  - Design of the system database;
  - Design of the logical processing components of Web GIS application;
  - Design of the user interface of the Web GIS application.
2 Requirements for GIS-based web application

The main goal of T6.2 is development of a Web GIS application:

- Using Open source software (eg. QGIS, Javascript GIS library, PostgreSQL and PostGIS extension, web framework).
- Using managed data via a GIS-based tool, especially targeting transport (infrastructure) planners.
- The tool must be tailored to strategic (transport, maintenance) planning needs.

Software processing will extract the underlying information in order to:

- Extend the availability of good quality asset management data,
- Help and determine the transport and commercial revenue potentials.

2.1 Requirements for Web GIS application

The Web GIS application will operate at the Data Centre level (the application server shall be hosted by UIC), and consider the following requirements and functional constraints (see Figure 2-1, sections user, map, GIS side):

- The user will be able to visualize the railway network.
- The user will be able to visualize the basemap of Open Street Maps (geographical overview).
- The user will have an optimal working scale at 1:20 000 000 in the WGS84 coordinate system.
- The user will be able to zoom and moving on the map with the mouse.
- The user will be able to filter by area selection (zoom) on the map, by means of screen limit (boundary).
- The user will be able to filter / select / inspect entity of the layers he considers relevant (administrative regions at ISO or NUTS level, railway stations, land use, population etc.)
2.2 Requirements for quality attributes of systems

The design of the system will be focused on the quality attributes thinking around critical problems that the design should solve.

The quality attributes are addressed during design rather than during development in order to improve the likelihood that the system will be successful in the long term.

The quality attributes considered in the design of the system are presented below:

- **Availability** - defines the proportion of time that the system is functional and is measured as a percentage of the total system downtime over a predefined period. Availability is affected by system errors, infrastructure problems, malicious attacks, and system load.
- **Conceptual Integrity** - defines the consistency and coherence of the overall design and includes the way that components are designed, as well as factors such as coding style and variable naming.
- **Flexibility** - system flexibility is the ability to adapt to varying environments and situations, and to cope with changes to business policies and rules. It is easy to reconfigure or adapt in response to different user requirements.
- **Interoperability** - System Interoperability is the ability of its diverse components to operate successfully by exchanging information by using services. Communication protocols, interfaces, and data formats are the key considerations for interoperability.
- **Maintainability** - System maintainability is the ability to undergo changes to its components, services, features, and interfaces as may be required when adding or changing the functionality, fixing errors, and meeting new business requirements. Maintainability can be measured in terms of the time it takes to restore the system to its operational status following a failure or removal from operation for upgrading. Improving system maintainability will increase efficiency and reduce run-time defects.
- **Manageability** - System manageability defines how easy it is to manage the application, through tools used in monitoring system and for debugging and performance tuning.
- **Performance** - is an indication of the responsiveness of the system to execute any action within a given interval of time. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within given amount of time. Factors affecting system performance include the demand for a specific action and the system’s response to the demand.
- **Reliability** - System reliability is the ability of the system to remain operational over time and is measured as the probability that the system will not fail to perform its intended functions over a specified interval of time. Improving the reliability of the system may lead to a more secure system because it helps to prevent the types of failures that a malicious user may exploit.
- **Scalability** - System scalability is the ability to function well when there are changes to the load or demand. The system will be able to be extended by scaling up the performance of the server. Typically, the system should be able to handle increases in size or volume. The aim is to maintain the system’s availability, reliability, and performance even when the load increases.
• **Security** - System security is the way that the system is protected from disclosure or loss of information and prevents unauthorized modification of information. The factors affecting system security are confidentiality, integrity, and availability. Authentication, encryption, and auditing and logging are the features used for securing systems.

• **Supportability** - System supportability defines how easy it is for operators, developers, and users to understand and use the system, and how easy it is to resolve errors when the system fails to work correctly. Supportability is the ability to provide support to the system when it fails to work correctly.

• **Testability** - System testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Testability allows the isolation of the faults in the system in a timely and effective manner.

• **Usability** - System usability defines how well the system meets the requirements of the user by being intuitive, easy to localize and globalize, able to provide good access for disabled users and a good overall user experience.

### 3 Purposes of system for GIS-based web application

In this task, from the technical point of view, we propose the implementation of GIS-based web application support for managed data mainly targets transport (infrastructure) planning.

#### 3.1 Global architecture of system

To develop this GIS-based web application, we focused on several architectural concepts and technologies to get a reliable, scalable, and responsive application (see Figure 3-1).

![Figure 3-1 Global architecture of system](image-url)
Designing this architecture took into account the specific business requirements (described in the section 2), and also gives possibility to easily combine only individual components in an existing IT infrastructure.

- **User Interface**

For user interface which operate equally well on handheld devices and on desktop browser we used modern front-end frameworks such as **JQuery** and **Bootstrap** together with a mapping component - **Leaflet**, which all can use a variety of background map sources like OSM, CartoDB and ArcGIS World Imagery.

- **Leaflet** - right now is the most popular javascript map library, with many available plugins. It is more mature than OpenLayers 3, so right now it might be a better choice than OpenLayers 3 for web mapping projects.
- **Bootstrap** - is an open source toolkit for developing with HTML, CSS, and JS. Also gives an easy way to create nice and responsive designs for any screen size.
- **JQuery** - is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

For the desktop, in order to harmonize geographic layers with our project requirements, we used the QGIS application. QGIS, at the moment, it is the most popular open source Desktop GIS, so it contains many plugins for almost every GIS functionality we'll need. Also, PostGIS DB tables can be created and published directly from QGIS interface.

- **Application server**

For our business-oriented system with a lot of custom functionalities, and to offer adequate flexibility for the future, it is best to set business logic outside of Geographic DB Server (e.g. PostgreSQL with PostGIS extension) and develop our own web application framework based on Django Framework.

- **Django Framework** - is an opens source high-level Python Web framework that encourages rapid development and clean, pragmatic design.

In our application server, we used multiple libraries and tools. All libraries mentioned below are already integrated in our application server, in the core or with help of plugins. We use these libraries also separately to fulfill specific business needs.

- **Django-Leaflet** – allows us to use Leaflet java script library in our Django projects.
- **Django-GeoJSON** – is a set of tools to manipulate GeoJSON data with Django:
  - (De)Serializer for (Geo)Django objects, querysets and lists
  - Base views to serve GeoJSON map layers from models
  - GeoJSON model and form fields to avoid spatial database backends (compatible with django-leaflet for map widgets)
- **Django-Redis** - is a full featured Redis cache/session backend for Django

- **Data sources**
The choice between traditional relational database and non-relational (NoSQL) database in the GIS world is the same as anywhere else, although at the moment there are more geospatial capabilities available in relational DBs then in NoSQL DBs. In our project we prefer the more mature relational DB approach, but would consider NoSQL in cases where data volume could grow unexpectedly very large, could expect thousands of requests in a second or should store data/documents having a variety of structures which would be hard to store in one specific schema.

- **PostgreSQL with PostGIS extension** - is the clear leader in open source geospatial databases. PostGIS is fast, has a lot of GIS features, and can be used from QGIS.
- **Redis Server** - Redis is an open source server, in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperlogs and geospatial indexes with radius queries. In our project, this server is very useful for storing the results of spatial queries, and reusing them without stressing the database server.

### 3.1.1 Database architecture

The database for the decision support tools consists of the following tables (see Figure 3-2):

- **gis_layers** - table responsible for the management of the types of layers used in our GIS system
- **gis_administrative_region** - table responsible for the management of the geographic administrative region (countries and regions)
- **gis_rail_network** - table responsible for the management of the geographic rail network at the macro and micro levels
- **gis_rail_station** - table responsible for the management of the geographic positions of rail stations (used at micro levels of rail network)
- **gis_rail_in_region** - table responsible for ensuring the link between the railway network layers and the administrative regions' layers
- **gis_rail_station_in_region** - table responsible for ensuring the link between the rail stations layers and the administrative regions' layers
- **gis_building** - table responsible for the management of the geographic contour of building relevant to the purpose of our project (eg train station, warehouse, industrial, university etc.)
- **gis_landuse** - table responsible for the management of the geographic contour residential areas, industrial areas
- **gis_places** - table responsible for the management of the geographic contour for cities, towns etc.
- **gis_population** – table responsible for management of the geographic representation density of population at NUTS level

There are six types of geographic for each layer, namely:

- Point
- MultiPoint
- MultiLineString
- Polygon
- MultiPolygon
Spatial reference system for all types of geometries is SRID 4326, and for graphic representation, WSG84 is used.
Figure 3-2 Database structure
3.1.2 NeTIRail GIS-based web application system architecture

Our web system for representing geographic data has a 2-level architecture:

- business logic and data access layer
- client application layer

All these levels have been developed and built through the Django framework (1). Specifically, this framework allowed us to build a system based on a modern MTV (Model-Template-View, see Figure 3-3) web architecture (2):

- **Model** - the data access layer. This layer contains anything and everything about the data: how to access it, how to validate it, which behaviours it has, and the relationships between the data.
- **Template** - the presentation layer. This layer contains presentation-related decisions: how something should be displayed on a Web page or other type of document.
- **View** - the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template(s). It is the bridge between models and templates.

![Figure 3-3 Django - MVT Architecture](image)

In order to achieve our task goals in manipulating geographic data, we had to integrate more software packages into Django framework (see Figure 3-4), namely:

- **Django psycopg2** – is a PostgreSQL + PostGIS adapter for Python
- **Geo Django** - is an module for Django that turns it into a world-class geographic Web framework. GeoDjango strives to make to create geographic Web applications. Its features include:
  - Extensions to Django’s ORM for querying and manipulating spatial data;
  - Loosely-coupled, high-level Python interfaces for GIS geometry operations and data manipulation in different formats.
- **Django REST-framework** - is a powerful and flexible toolkit for building Web APIs based on REST architecture (3). The Django REST Framework provides powerful model serialization,
display data using standard function-based views, or get granular with powerful class-based views for more complex functionality.

- **Django REST-framework-GIS** - provides geographic addons for django REST framework such as a GeometryField field and a GeoJSON\(^1\) serializer.

- **Django GeoJSON** - is a set of tools to manipulate GeoJSON with Django:
  - (De)Serializer for (Geo)Django objects, querysets and lists etc.
  - Base views to serve GeoJSON map layers from models
  - GeoJSON model and form fields to avoid spatial database backends

- **Django Redis Cache** - is a cache backend for the Django web framework. It uses the redis server, which is a memory key-value data structure server.

- **Django Leaflet** - is a set of tools to allows us to use Leaflet javascript library in Django web framework

---

\(^1\) GeoJSON - is a javascript data format for encoding a variety of geographic data structures. GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon. Geometric objects with additional properties are Feature objects. Sets of features are contained by FeatureCollection objects.
3.1.2.1 Business logic and access data layer

This layer of our Web GIS system works at the server-side level, and is implemented on a REST API architecture. This component is responsible for solving the following issues:

- For implementation in ORM model format of relational GIS data base (see Figure 3-5).

![Figure 3-5 Python class diagram of data model implemented in our GIS-based web system](image)
This model is deploying our database structures within the Django Framework. In the Django framework, the model is the only definitive source of information about the data managed in the application. It contains the essential fields and behaviours of the data we store. In general, each class of model correspond to a single table of the database. That’s why our implementation model in Python has the same structure as our database (see Figure 3-2).

Basically, this model is the basis for the business logic level (BLL) where data is processed as needed. BLL items are listed below.

- For providing access and delivering data in GeoJSON format (see Figure 3-6)
• For providing operations to handle geographic data and associated metadata (Create, Read, Update and Delete)

![Class Diagram for operations to handle geographic data and associated metadata](image1)

Figure 3-7 Class Diagram for operations to handle geographic data and associated metadata

• For creating query data based on dynamic filters

![Class diagram for creating query data based on dynamic filters](image2)

Figure 3-8 Class diagram for creating query data based on dynamic filters
3.1.2.2 Client-side application layer

This module provides the interface between the user and Business logic and access data layer of our system.

For the user interface that works on the desktop browser, we used modern front-end frameworks such as JQuery and Bootstrap together with a mapping component - Leaflet, which all can use a variety of background map sources like OSM, CartoDB and ArcGIS World Imagery.

In order to achieve our task goals in manipulating geographic data by means of the user interface, it was necessary to integrate all these components into the Django Framework, in order to be able to interact with the GIS Rest component described in section 3.1.2.1.

Moreover, this component has been designed to work in AJAX mode. With AJAX, our Web GIS application can send and retrieve data from a GIS Rest component asynchronously (in the background) without interfering with the display and behaviour of the existing page. By decoupling the data interchange layer from the presentation layer, AJAX allows Web application pages to change content dynamically without the need to reload the entire page.

To achieve a common interface template, we have integrated the Bootstrap library into the Django framework and created our own theme (via CSS, see Figure 3-9 and Figure 3-10) (4).

![Figure 3-9 Common form template based on CSS](image-url)
To manipulate interface elements in an AJAX manner, we have integrated the JQuery library (5).

The most important integrated library in our application is Leaflet (6). This javascript library enabled us to build dynamic maps as well as implement controls for the selection and display of geographic layers, to implement localization, zooming and dragging map functions (see Figure 3-12).

[Figure 3-10 Integration diagram of Bootstrap library in our GIS application]

[Figure 3-11 Integration diagram of JQuery library in our GIS application]
3.2 NeTIRail GIS-based web application

3.2.1 Geographic layer

Within our GIS we handle a multitude of geographic information in vector format and structured in layers. This information in vector format together with specific metadata is stored in separate tables of our database. Geographic information is partly constituted by OpenStreetMap geographic data. The database structure of geographic layer is strongly inspired by OpenStreetMap model.

The `gis_layers` table is responsible for the management of the types of layers used in our GIS system.

In this table, we manage the following categories of information:
• code – unique code for each type of layer managed
• layer – name of specific layer
• descriptions - contains descriptive information on the layer type

The types of layers managed in our database are shown in the following table.

<table>
<thead>
<tr>
<th>code</th>
<th>layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>city</td>
<td>City outlines(^2)</td>
</tr>
<tr>
<td>1002</td>
<td>town</td>
<td>Town outlines(^3)</td>
</tr>
<tr>
<td>1003</td>
<td>village</td>
<td>Village outline(^4)</td>
</tr>
<tr>
<td>1010</td>
<td>suburb</td>
<td>Suburb outline(^5)</td>
</tr>
<tr>
<td>1041</td>
<td>country</td>
<td>A county outline</td>
</tr>
<tr>
<td>1040</td>
<td>region</td>
<td>A region outlines</td>
</tr>
<tr>
<td>1050</td>
<td>locality</td>
<td>Locality outline</td>
</tr>
<tr>
<td>1500</td>
<td>building</td>
<td>Building outlines</td>
</tr>
<tr>
<td>5601</td>
<td>railway station</td>
<td>A larger railway station of mainline rail services.</td>
</tr>
<tr>
<td>5602</td>
<td>railway halt</td>
<td>A smaller, local railway station, or subway station.</td>
</tr>
<tr>
<td>5603</td>
<td>tram stop</td>
<td>A tram stops.</td>
</tr>
<tr>
<td>6101</td>
<td>rail</td>
<td>Regular railway tracks.</td>
</tr>
<tr>
<td>6106</td>
<td>narrow gauge</td>
<td>A narrow-gauge railway track.</td>
</tr>
<tr>
<td>6103</td>
<td>subway</td>
<td>Underground railway tracks. OSM Tags railway=subway</td>
</tr>
<tr>
<td>6104</td>
<td>tram</td>
<td>Tram tracks (may be incident with roads).</td>
</tr>
<tr>
<td>6105</td>
<td>monorail</td>
<td>A monorail track.</td>
</tr>
<tr>
<td>6107</td>
<td>miniature</td>
<td>A miniature railway track.</td>
</tr>
<tr>
<td>6102</td>
<td>light rail</td>
<td>Light railway tracks, often commuter railways.</td>
</tr>
<tr>
<td>7203</td>
<td>residential</td>
<td>A residential area.</td>
</tr>
<tr>
<td>7204</td>
<td>industrial</td>
<td>An industrial area.</td>
</tr>
<tr>
<td>7209</td>
<td>commercial</td>
<td>A commercial area.</td>
</tr>
<tr>
<td>9999</td>
<td>population</td>
<td>Population Density at NUTS level.</td>
</tr>
</tbody>
</table>

Table 3-1 The types of layers managed in geographic database

3.2.1.1 Administrative Regions layer
The information, related to Administrative regions, in vector format together with specific metadata are stored in table gis_administrative_region.

The gis_administrative_region table is responsible for the management of the geographic administrative region (countries and regions)

In this table, we manage the following categories of information:
- id - unique identifier (auto increment)

---

\(^2\) In OpenStreetMap data, city is used for identify the largest settlement or settlements within the territory, including national, state and provincial capitals and other major conurbations.

\(^3\) In OpenStreetMap data, town is used for identify an important urban center that is larger than a village smaller than a city and not a suburb. Towns normally have a good range of commercial activities used by people nearby the village.

\(^4\) In OpenStreetMap data, village is used for identify a settlement between 1 000 and 10 000 inhabitant. Hence it may vary by country.

\(^5\) In OpenStreetMap data, suburb is used for identify a major area in a town or city with a distinct and a recognized local name and identity. Suburb may have uncertain boundaries, may overlap with other suburbs.
- **osm_id** - OSM Id taken from the Id of this feature in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique, because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **name** - name of this feature (country or region)
- **local_name** – local name of this feature (country or region)
- **iso1** – code of country according to standard **ISO 3166-1**. This field will be used at both micro and macro levels
- **iso2** - code of region according to standard **ISO 3166-2**. This field will only be used at micro level.
- **admin_lvl** – this field represent numbers of administrative divisions used by countries and their major dependent territories.
- **geom** - this field represent the geographic geometry of country respective region of country. This field are in format (geometry::multipolygon, srid::4326)

At this time in the database we have entered the data for 47 countries (the European continent) see Figure 3-14, namely: Albania, Andorra, Austria, Belarus, Belgium, Bosnia and Herzegovina, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Georgia, Germany, Gibraltar, Greece, Hungary, Ireland, Italy, Kosovo, Latvia, Liechtenstein, Lithuania, Luxembourg, Macedonia, Malta, Metropolitan France (only the part of the European continent), Moldova, Monaco, Montenegro, Norway, Poland, Portugal (only the part of the European continent), Romania, San Marino, Serbia, Slovakia, Slovenia, Spain (only the part of the European continent), Sweden, Switzerland, The Netherlands (only the part of the European continent), Turkey, Ukraine, United Kingdom (only the part of the European continent), Vatican City

---

6 this field was attached in view of a possible connection with OpenStreetMap.
Regarding the regions at this time in the database we have entered the data for 566 regions for each country (see Figure 3-15) less for Andorra, Liechtenstein, Montenegro, Vatican City (for these contours of regions they correspond to the country) and Moldova (this is under construction).

3.2.1.2 Railway Network layer
This layer alongside the station layer is the most important because all the metadata (RINF, MERITS, RTM data, etc.) that will be managed in Task 6.3 will directly interact with these layers.
The information, related to Rail Network at macro and micro level, in vector format together with specific metadata are stored in table `gis_rail_network`.

The `gis_rail_network` table is responsible for the management of the geographic rail network at the macro and micro levels.

In this table, we manage the following categories of information:

- **Id** - unique identifier (auto increment)
- **osm_id** - OSM Id taken from the Id of this feature (node_id “stations in our case”, way_id “railway in our case”, or relation_id) in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - Class name of this feature (identical with the name of the layer)\(^8\). This does not add any information that is not already in the "code" field but it is better readable.
- **name** - name of this feature (name of part rail route, e.g. “Masonbachaquädukt (Überwölbter Einschnitt)"
- **geom** - this field represent the geographic geometry railway at macro or micro level. This field are in format (geometry::multilinestring, srid::4326)
- **data_level** - this field represent the level of geometry (macro or micro level)

At this time in the database we have entered the data for full macro level (10869 geometry elements) and 99% of data for micro level (989469 geometry elements) over the EU continent. The source of geographic information is Geofabrik\(^9\)

---

\(^7\) this field was attached to a possible connection with OpenStreetMap..

\(^8\) This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database

\(^9\) Geofabrik - [https://www.geofabrik.de/geofabrik/](https://www.geofabrik.de/geofabrik/)
3.2.1.3 Rail Stations layer

The information, related to Rail Stations, in vector format together with specific metadata are stored in table `gis_rail_station`.

The `gis_rail_station` table is responsible for the management of the geographic positions of rail stations (used at micro levels of rail network).

In this table, we manage the following categories of information:
- **Id** - unique identifier (auto increment)
- **osm_id**\(^{10}\) - OSM Id taken from the Id of this feature (node_id “stations in our case”, way_id “railway in our case”, or relation_id) in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - Class name of this feature (are identical to name of layer eg. Rail station or rail halt)\(^{11}\). This does not add any information that is not already in the "code" field but it is better readable.
- **name** - name of this rail station or rail halt (name of part rail route, e.g. Bartolomeu)
- **geom** - this field represent the geographic geometry rail stations at micro level. This field are in format (geometry::point, srid::4326)

At this time in the database, we have entered the data for 52 284 items (32 210 rail stations and 20 074 rail halts). The source of geographic information is [Geofabrik]\(^{12}\)

![Figure 3-18 Example of layer Rail Stations at micro level in region Ile de France](image)

### 3.2.1.4 Population layer

The information, related to EU Population, in vector format together with specific metadata are stored in table **gis_population**.

The **gis_population** table is responsible for the management of 2016 population information for NUTS level.

\(^{10}\) this field was attached to a possible connection with OpenStreetMap.

\(^{11}\) This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database

\(^{12}\) Geofabrik - [https://www.geofabrik.de/geofabrik/](https://www.geofabrik.de/geofabrik/)
In this table, we manage the following categories of information:

- **id** - unique identifier (auto increment)
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - class name of this feature (are identical to name of layer, e.g. rail station or rail halt)$^{13}$ . This does not add any information that is not already in the "code" field but it is better readable.
- **nuts** - acronym of the NUTS region (1,2,3)
- **nuts0** - acronym of the country
- **name** - name of the NUTS region
- **Population** - total population for NUTS region
  - **males** – total males population for NUTS region
  - **females** – total female population for NUTS region
- **households** – number of households for the NUTS region
- **pers_hh** - average number of persons per household for the NUTS region
- **pop_growth** - population growth rate for NUTS region
- **pop_densit** - population density per square kilometre for NUTS region
- **dwellings** – number of dwellings for NUTS region
- **age_0_14** – total population age between 0 and 14 for NUTS region
- **age_15_29** – total population age between 15 and 29 for NUTS region
- **age_30_44** – total population age between 30 and 44 for NUTS region
- **age_45_59** – total population age between 45 and 59 for NUTS region
- **age_60_** – total population age 60 and over for NUTS region
- **pp_incntry** - yearly amount of purchasing power (i.e. disposable income) for the NUTS region in Euro
- **pp_categor** - yearly amount of purchasing power (i.e. disposable income) for the NUTS region in Euro
- **pp_growth** - purchasing power growth rate for the NUTS region in percent ranges.
- **gdp** - gross domestic product by NUTS region
- **employed** – number of employed persons for NUTS region
- **sqkm** - area of NUTS region in square kilometres
- **shape_len** - length of geographic limit according to geographic object
- **shape_area** - area of geographic limit according to geographic object
- **min_zoom** - accepted minimum zoom level of each administrative limit. This field helps to illustrate the geometric object of the NUTS region on screen according to level of the main map
- **max_zoom** - accepted maximum zoom level of any NUTS limit. This field helps to illustrate the geometric object NUTS region on screen according to level of the main map

At this time in the database NUTS classifications contains, European counties NUTS 0, 98 regions at NUTS 1, 1,276 regions at NUTS 2 and 1342 regions at NUTS 3 level.$^{14}$ (The source of geographic information is ArcGIS$^{15}$)

---

$^{13}$ This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database

$^{14}$ [http://ec.europa.eu/eurostat/web/nuts/overview](http://ec.europa.eu/eurostat/web/nuts/overview)

$^{15}$ ArcGIS - Europe Population Density - [https://www.arcgis.com/home/item.html?id=cf3c8303e85748b5bc097cddb5d39c31](https://www.arcgis.com/home/item.html?id=cf3c8303e85748b5bc097cddb5d39c31)
- NUTS 0\textsuperscript{16}: administrative country boundaries
- NUTS 1: major socio-economic regions
- NUTS 2: basic regions for the application of regional policies

NUTS 3: small regions for specific diagnoses

\footnotesize{\textsuperscript{16} NUTS 0 level is an added level for a better coherence in population table.}
Figure 3-20 Example of layer population at level NUTS 1

Figure 3-21 Example of layer population at level NUTS 2
3.2.1.5  Land Use Layer

The information, related to human use of land at EU level, in vector format together with specific metadata are stored in table `gis_landuse`.

The `gis_landuse` table is responsible for the management of the human use of land.

In this table, we manage the following categories of information:

- **id** - unique identifier (auto increment)
- **osm_id**\(^{17}\) - OSM Id taken from the Id of this feature in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - Class name of this feature (are identical to name of layer eg. Commercial, residential, industrial)\(^{18}\). This does not add any information that is not already in the "code" field but it is better readable.
- **name** - name of this rail station or rail halt (name of commercial, industrial or residential area etc. eg. Pyce )
- **geom** - this field represent the geographic geometry places. This field are in format (geometry::multipolygon, srid::4326)

---

\(^{17}\) this field was attached to a possible connection with OpenStreetMap.

\(^{18}\) This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database.
3.2.1.6 Places layer

The information, related to illustration of significant human settlements at EU level, in vector format together with specific metadata are stored in table *gis_places*.

The *gis_places* table is responsible for the management of illustrate significant human settlements (city, town, suburb, village etc.).

In this table, we manage the following categories of information:

- **Id** - unique identifier (auto increment)
- **osm_id**\(^{19}\) - OSM Id taken from the Id of this feature in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - Class name of this feature (are identical to name of layer eg. City, town, suburban)\(^ {20}\). This does not add any information that is not already in the "code" field but it is better readable.
- **name** - name of this rail station or rail halt (name of the city, town, village etc. eg. Pyce)
- **geom** - this field represent the geographic geometry places. This field are in format (geometry::multipolygon, srid::4326)

\(^{19}\) this field was attached to a possible connection with OpenStreetMap.

\(^{20}\) This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database.
3.2.1.7 Land Use Layer

The information, related to illustration of any kind of building, in vector format together with specific metadata are stored in table `gis_buildings`.

The `gis_buildings` table is responsible for the management of any kind of building, such as houses, factories, destroyed building etc..

In this table, we manage the following categories of information:

- **Id** - unique identifier (auto increment)
- **osm_id**\(^{21}\) - OSM Id taken from the Id of this feature in the OSM database. In case several features in the OSM database are joined into one feature, this is one of the Ids. This Id is not necessarily unique because one OSM object can result in several geometry objects.
- **code** - provides the link between the current record and the layer category it belongs to
- **fclass** - Class name of this feature (are identical to name of building eg. appartement, hospital, mall etc.)\(^{22}\). This does not add any information that is not already in the "code" field but it is better readable.
- **name** - name of this building
- **geom** - this field represent the geographic geometry places. This field are in format (geometry::multipolygon, srid::4326)

---

\(^{21}\) this field was attached to a possible connection with OpenStreetMap. We will give up if it will not be used in the future.

\(^{22}\) This field is used only in development phase for verified correct layer classification in import data process by different sources. This field will be deleted in final form of database.
3.2.2 NeTIRail GIS-based web functionality

The NeTIRail GIS-based web is the core component for developing the Decision Support Tools, which will be developed under Task T6.3.

NeTIRail GIS-based web provides the basic features of a GIS application by providing user functions for

- displaying thematic layers (rail network, rail stations, land use, population densities, places etc.) (see section 3.2.1)
- filtering layers
- display layer metadata
- zoom in / zoom out functions

Moreover, through this component, ReST can manage (create, modify, delete) all geographic information.

For filtering and displaying data efficiently, we created dynamic filtering functions based on identifying the coordinates of the map part displayed (bounding box of map) and extracting from the GIS database only data related to the displayed map bounding box.

Because points of interest can exceed 50000 records, we have created a mechanism by which they can be grouped and clustered, and by mouse clicks can be unclustered and displayed at optimal resolution (zoom level).
The most important print screens of our web GIS interface are shown below.

**Figure 3-26 Rail network representation at EU level**

**Figure 3-27 Example of clustered Station layer**
Also, at our ReST GIS API we implemented a ReST API interface in order to manage GIS data via the web interface. The most important print screens of our ReST API interface are shown below:

Figure 3-29 The GIS ReST API interface for accessing principal functions for the manipulation of GIS data
Figure 3-30 Example of ReST API interface for Get (a) and Post (b) data

Figure 3-31 Example of ReST API interface for Put and Delete data
4 Testing of GIS-based web application

The overall objective of the testing is to identify errors, defects and shortcomings in the operation and the integrity of the all components of system software, and trigger review procedures for the relevant functions and components, in terms of re-design and corrective implementation.

Our testing process are separates the whole process in two Steps:

1) Step I – The internal testing of code quality and functionality, it includes
   - Unit Tests
   - Component Tests
2) Step II – It contains test cases that are business driven and are implemented to support the team. This step focuses on the requirements. The kind of test performed in this phase is
   - Testing of examples of possible scenarios and workflows

4.1 Unit Tests and Component Tests

In the testing process conducted over the development of the application all the functional requirements presented in Section 2.

The tester, who is conducting the testing, follows specific steps like:

- Performing of the activities or tests that are described in the test cases;
- Recording of the test results in the test case execution logs;
- Recording of any errors in the error / defect logs.

The overall testing map is presented below:

![Overall Testing Map](image)

**Figure 4-1 The Overall Testing Map**

A. Pass/Fail criteria

When conducting a test case, the result may be a success or a failure. The recording of the results is made in the test case execution log form.

For each identified error, a severity level is assigned, according to the following table:
**D6.3 – GIS-based web application**

**NeTIRail-INFRA**

H2020-MG-2015-2015 GA-636237

2018/02/13

<table>
<thead>
<tr>
<th>Severity level</th>
<th>Effect on the tested function</th>
<th>Effect on the overall system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Incomplete or incorrect feature, problematic implementation.</td>
<td>Incomplete operation. The whole subsystem or parts of it are not working properly or “crash”.</td>
</tr>
<tr>
<td>2</td>
<td>Non-critical or unimportant omission or insignificant error in subsystem’s operation.</td>
<td>Insignificant effect in the subsystem’s overall performance.</td>
</tr>
<tr>
<td>3</td>
<td>“Decorative” error (e.g. image, word, size, etc.)</td>
<td>Effect only on the usability of the subsystem, not on its functionality.</td>
</tr>
</tbody>
</table>

Table 4.1 Error severity level

The acceptable thresholds for the reliable operation of the system are the following:

- There is no problem of severity level “1”.
- There are less than 4 errors of severity level “2”.
- There are no more than 8 errors of severity level “3”.

**B. Traceability matrix**

The following table presents the correspondence between the identified system functional specifications and the test cases:

<table>
<thead>
<tr>
<th>Specification code</th>
<th>Specification title</th>
<th>Test case code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Create data GeoJSON format for each specific layer</td>
<td>TC1</td>
</tr>
<tr>
<td>S2</td>
<td>Storage of geographic data collected into local database</td>
<td>TC2</td>
</tr>
<tr>
<td>S3</td>
<td>Transmission data via ReST API the web GIS module</td>
<td>TC3</td>
</tr>
<tr>
<td>S4</td>
<td>Interrogation and graphical display into web GIS module</td>
<td>TC4</td>
</tr>
</tbody>
</table>

Table 4.2 List of test case specification

**C. Test cases**

<table>
<thead>
<tr>
<th>TEST CASE 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>n.</th>
<th>TC1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Create data GeoJSON format for each specific layer</td>
</tr>
<tr>
<td>Use environment</td>
<td>PyCharm Professional 2017.3, Web Browser</td>
</tr>
</tbody>
</table>

**Brief description**

**Scope**

- Returns all data in GeoJSON format
- Return all metadata for each layer type (according to data model definitions, see Section 3.1.2.1)

**Function**

- get AdministrativeRegion()
- get RailNetwork()
- get RailStation()
- get Buildings()
- get Landuse()
- get Places()
### TEST CASE 2

<table>
<thead>
<tr>
<th>n.</th>
<th>TC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Storage of geographic data collected into local database</td>
</tr>
<tr>
<td>Use environment</td>
<td>PyCharm Professional 2017.3, Web Browser</td>
</tr>
</tbody>
</table>

**Brief description**

**Scope**
- Verify right of application to write data into database
- Write data via Rest API into database

**Function**
- Get, Post, Put, Delete AdministrativeRegion()
- Get, Post, Put, Delete RailNetwork()
- Get, Post, Put, Delete RailStation()
- Get, Post, Put, Delete Buildings()
- Get, Post, Put, Delete Landuse()
- Get, Post, Put, Delete Places()
- Get, Post, Put, Delete Population()

**INPUT**
- HTTP Get, Post, Put, Delete
- checkPermissiosNeeded()

**OUTPUT**
- Geographic data and associated metadata

### TEST CASE 3

<table>
<thead>
<tr>
<th>n.</th>
<th>TC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Transmission data via Rest API the web GIS module</td>
</tr>
<tr>
<td>Use environment</td>
<td>PyCharm Professional 2017.3, Web Browser</td>
</tr>
</tbody>
</table>

**Brief description**

**Scope**
- Request data in GeoJSON format Rest API the web GIS module

**Function**
- get AdministrativeRegion()
- get RailNetwork()
- get RailStation()
- get Buildings()
- get Landuse()
- get Places()
- get Population()

**INPUT**
- HTTP Get

**OUTPUT**
- GeoJSON data (Geographic data and associated metadata)
TEST CASE 4

n. TC4

Title Interrogation and graphical display into web GIS module

Use environment PyCharm Professional 2017.3, Web Browser

Brief description

Scope
- Request data in GeoJSON format via AJAX query at web client module
- Display data in GeoJSON on web Gis map

Function
- Get, Post AdministrativeRegion()
- Get, Post RailNetwork()
- Get, Post RailStation()
- Get, Post Buildings()
- Get, Post Landuse()
- Get, Post Places()
- Get, Post Population()
- add Layer()
- remove Layer()

INPUT
- HTTP Get

OUTPUT
- GeoJSON data (Geographic data and associated metadata)
- Display Layer (based on received GeoJSON data) on the map

D. Test case execution log form

TEST CASE EXECUTION LOG FORM

GENERAL INFORMATION

Test cases TC1, TC2, TC3, TC4

Date 02/02/2018

TEST CASE EXECUTION LOG

<table>
<thead>
<tr>
<th>Test case n.</th>
<th>Pass/Fail</th>
<th>Specific testing feature</th>
<th>Error n.</th>
<th>Acceptance</th>
<th>Comments if not accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>Pass</td>
<td>• get AdministrativeRegion()</td>
<td>0</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get RailNetwork()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get RailStation()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get Buildings()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get Landuse()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get Places()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• get Population()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TC2</td>
<td>Pass</td>
<td>• Get, Post, Put, Delete AdministrativeRegion()</td>
<td>0</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Get, Post, Put, Delete RailNetwork()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Get, Post, Put, Delete RailStation()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Get, Post, Put, Delete Buildings()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TEST CASE EXECUTION LOG FORM

**GENERAL INFORMATION**

<table>
<thead>
<tr>
<th>Test cases</th>
<th>TC1, TC2, TC3, TC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>02/02/2018</td>
</tr>
</tbody>
</table>

**TEST CASE EXECUTION LOG**

<table>
<thead>
<tr>
<th>TC3</th>
<th>Pass</th>
<th>• Get, Post, Put, Delete Landuse()</th>
<th>• Get, Post, Put, Delete Places()</th>
<th>• Get, Post, Put, Delete Population()</th>
<th>0</th>
<th>Yes</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC4</td>
<td>Pass</td>
<td>• Get, Post AdministrativeRegion()</td>
<td>• Get, Post RailNetwork()</td>
<td>• Get, Post RailStation()</td>
<td>• Get, Post Buildings()</td>
<td>• Get, Post Landuse()</td>
<td>• Get, Post Places()</td>
</tr>
</tbody>
</table>

### 4.2 Prototype test in real conditions

The purpose of real-condition testing is to demonstrate the functionality and scalability of the system under the conditions required by the requirements of the functional specification.

Testing was performed on three web browsers and on tree PC with different configurations (one of the high class, the second in the middle class, and one of the lower class).

In the tests, the following visual and functional aspects were followed:

- Correct rendering of map control and layer on the map using different web browsers
- Correct filtering of data
- Response time and load time of GeoJSON information on the map

After testing activities mentioned above were not found to malfunction or block the system.

The correct rendering of map control and layer on the map using different web browser is shown in the figures below.
The following are the significant response time diagram for GeoJSON data exceed 50000 records (GeoJSON for all Rail Station and Halts) and over 10MB dimension.

This is a non-filtered query result, where transforming data in GeoJSON + data transfer + local porting was performed in 42 seconds.
This is a filtered query result by a specific type of Rail Stations, where portability + data transfer + local porting was performed in 15 seconds.

But our system is designed to generate dynamic map-level map filters, resulting in a small number of results that are processed and transferred in less than 1 second (see Figure 4-5). The above tests are just stress tests of our system.
5 Conclusions

This deliverable has presented the system, which was developed for the Task 6.2.

The standards and the methodology for assessing the comfort were also presented within the deliverable. This methodology has been used in design and development web software procedures.

Our web system for representation geographic data is constituted in the architecture based on two levels, namely:

- business logic and access data layer
- client application layer

All these levels have been developed and built through the Django framework. Specifically, this framework allowed us to build a system based on a modern MTV (Model-Template-View) web architecture.

Our Systems testing was conducted in three sub activities, namely:

- Defining of the test plan according to the technical specifications
- Unit Tests and Component Tests
- Prototype test in real conditions

During the testing activities have not found malfunction or blockage of the system.

The NeTIRail GIS-based web is the core component for developing the Decision Support Tools, which will be developed under Task T6.3.

In the next period, all managed geographic data and result layers will directly interact with specific Rail metadata (RINF, MERITS, RTM data, etc.) which will be managed through the software components that will be developed in Task T6.3
6 References


