Deliverable D6.4
Decision support tools for implementation of technologies
Submission date: 17 June 2018
**Lead contractor**

UIC

**Contributors**

ADS

**Project Coordinator**

University of Sheffield, USFD
Executive Summary

The decision support tool has been developed as a Web application. Since the innovative technologies developed in other NeTIRail work packages all relate to railway infrastructure and their characteristics, the pitfall of unaffordable data acquisition and update had to be avoided.

This was achieved by obtaining network characteristics from the public European Register of Infrastructure (RINF), and network representation from the OpenStreetMap project that come under a free, open license. Moreover, both datasets are associated “on the fly” by the application, thus being able to be updated independently from each other.

RINF does not carry characteristics that are essential for impact assessment, such as train loads or economic information. Consequently, the application user is allowed to add such characteristics (parameter names and values) to the network description.

Finally, sections of lines can be associated with cost or benefit functions related to the technologies, allowing the generalization of the cost-benefit analyses proposed under WP2, 3, and 4 to arbitrary, user-selected sets of lines. These functions can be input by the user, instead of being hard-coded. Functions are proposed in subsequent deliverables, D6.5 and D6.6.

Given the above, the decision support tool is functionally rich, extensible at will, and resting on permanently refreshed, available free data.

To further future-proof the decision support tool, we decided to adopt effective, but “mainstream” software architecture principles (REST...), web development frameworks (DJANGO...), and relational database (PostgreSQL).

Even more importantly, the decision support tool is largely hardware-independent, as it can be used by any client using any of the current web browsers (Firefox, Edge, Safari...) on most devices and operating systems.

The above precautions were not taken in anticipation of a large user base: the decision support tool remains somewhat esoteric in its advanced features. However, the technology and data choices were made to allow a quick development, and to maximize the likelihood of further application support and maintenance with limited resources.

The NeTIRail-INFRA Grant Agreement describes the decision making tool as supporting analysis beyond the existing network and geography, which implies multilingual interface, support for “what if” scenarios and ability to import data. The tool deviates from the grant agreement in that the interface has not been translated, however, as demonstrated in this deliverable, the web based tool does allow localization parameters to be set, and allows for different languages to be used in the interface. The ability to import data and run “what if” scenarios have been fully implemented. The tool also allows custom formulae to be entered, which allows for much greater customization and give end users the flexibility to use the tool for a much wider range of purposes.
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>3</td>
</tr>
<tr>
<td>Table of contents</td>
<td>4</td>
</tr>
<tr>
<td>Abbreviations and acronyms</td>
<td>6</td>
</tr>
<tr>
<td>1 Final Requirements for GIS-based web application</td>
<td>7</td>
</tr>
<tr>
<td>1.1 Presentation of the GIS-based web application</td>
<td>7</td>
</tr>
<tr>
<td>1.2 Specifications of the GIS-based web application (User Story)</td>
<td>7</td>
</tr>
<tr>
<td>2 RINF data</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Presentation of the RINF Data</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Set of Parameters exported from RINF</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Conclusion</td>
<td>19</td>
</tr>
<tr>
<td>3 Benefits calculation and decision-making aids</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Fundamental option: treat code like data</td>
<td>20</td>
</tr>
<tr>
<td>4 Infrastructure specification for host web-application</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Technical Architecture Document</td>
<td>21</td>
</tr>
<tr>
<td>4.1.1 Global architecture scheme</td>
<td>22</td>
</tr>
<tr>
<td>4.1.2 Technical platform</td>
<td>24</td>
</tr>
<tr>
<td>4.2 Class diagrams</td>
<td>26</td>
</tr>
<tr>
<td>4.2.1 GIS class diagrams</td>
<td>26</td>
</tr>
<tr>
<td>1.1.1 RINF class diagram</td>
<td>28</td>
</tr>
<tr>
<td>4.2.2 Client-side application class</td>
<td>29</td>
</tr>
<tr>
<td>4.3 Data Description</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1 Relational Database Structure</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1.1 Tables dedicates to GIS components</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1.2 Tables dedicates to RINF components</td>
<td>34</td>
</tr>
<tr>
<td>4.3.2 Physical Model</td>
<td>36</td>
</tr>
<tr>
<td>4.4 Code Description</td>
<td>38</td>
</tr>
<tr>
<td>4.4.1 Code architecture</td>
<td>38</td>
</tr>
<tr>
<td>4.4.2 Identification and authentication mechanisms</td>
<td>40</td>
</tr>
<tr>
<td>4.4.3 Importation of RINF data: mechanism</td>
<td>42</td>
</tr>
<tr>
<td>4.4.4 Architecture of the formula mechanism</td>
<td>44</td>
</tr>
<tr>
<td>4.4.5 ORM</td>
<td>45</td>
</tr>
<tr>
<td>4.4.6 Leaflet</td>
<td>46</td>
</tr>
</tbody>
</table>
4.4.7 GIS.......................................................................................................................... 49
4.4.8 Django REST.................................................................................................................. 51
4.4.9 Management of multiple languages................................................................. 54
4.4.10 Description of the application log process .................................................. 56
4.4.11 Description of the ERROR management process ..................................... 56
4.4.12 Libraries used ........................................................................................................ 56
4.5 HARDWARE PLATFORMS.............................................................................................. 59
  4.5.1 Recommended environment.............................................................................. 59
  4.5.2 Performance specifications................................................................................. 59
5 Conclusion...................................................................................................................... 60
6 References...................................................................................................................... 61
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation / Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRUD</td>
<td>Create, Read, Update and Delete</td>
</tr>
<tr>
<td>EUAR</td>
<td>European Union Agency for Railways</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MERITS</td>
<td>Multiple European Railways Integrated Timetable Storage</td>
</tr>
<tr>
<td>OP</td>
<td>Operational Point (in RINF)</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Mapping</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer; an architectural style for web services.</td>
</tr>
<tr>
<td>RINF</td>
<td>European Register of Infrastructure</td>
</tr>
<tr>
<td>SoL</td>
<td>Section of Line (in RINF)</td>
</tr>
<tr>
<td>TSI</td>
<td>Technical Specification for Interoperability</td>
</tr>
</tbody>
</table>
1 Final Requirements for GIS-based web application

1.1 Presentation of the GIS-based web application

Geographic Information Systems (GIS) and online web technology have made spatial information easily accessible, while providing powerful online mapping applications. Web GIS applications are expedient tools for mapping and visualizing data across a geographic view, allowing data to be shown as maps.

Today, mapping any type of data has become very popular, thus these tools improve productivity. As well as any type of user from other sectors, users in the rail sector may also use a Web GIS application for effective management or regulation. The aim of this work package is to create such a web application dedicated to railways, as a decision-supporting tool.

This Web Based GIS tool can help transport planners, infrastructure procurement, maintenance staff and also any type of user to visualize, analyze and share geographic information on a wide range of topics, with focus on railway infrastructure information.

1.2 Specifications of the GIS-based web application (User Story)

User stories have been formulated in order to develop the specifications which will enable the web application to play the case study line scenarios. The user stories consider the user’s point of view and aim to identify what the user should be able to accomplish with the tool and how they will interact with it. These user stories then form the basis for the specification.

The number, in front of the user stories, corresponds to their reference throughout all the documentation of the development project. It also gives information about the nesting of user stories among them.
1 Global Technical Story

1.1 Optimise the speed of GIS
1.2 JS file for User Interface
1.3 Redesign DB for user profile (done)
1.4 Install a server for web application
1.5 Put the application into production
1.6 Install Github
1.7 The applications must be able to work, on site, with several layers
1.8 Concerning RINF, define and implement an ORM model:
   18.1 For the model user-level privacy, options were provided
   18.2 For the model, there were possibilities to import and store binary files
   18.3 For the model, the transposition of imported data into relational format data files was taken into consideration
   18.4 Specific functions have been designed to convert extracted numerical data (latitude, longitude for points, or multipoint where applicable) into geographic format

2 GIS-API

2.1 REST Services: The web application will be able to access tools to filter GIS data for each layer
2.2 The admin will be able to manage (CRUD) layers
2.3 REST Services: The web application will be able to access tools to serialize layers
2.4 The admin will be able to manage the GIS-Data
3 Parameter and formula WEB-APPLICATION (Formula, RINF, NEW PARAMETERS for other GIS Web Application)

3.1 The admin user and application will be able to manage and access RINF data (SOL/OP)
   3.1.1 The admin user will be able to import RINF data (XML)
   3.1.2 The admin user will be able to manage (CRUD) RINF data
   3.1.3 The application will be able to transform a non-numeric value into a numerical list of values
   3.1.4 REST Services : The GIS web application will be able to access to RINF data
   3.1.5 REST Services : The GIS web application will be able to have permission to use Rest Services

3.2 The admin user and application will be able to manage and access NEW PARAMETERS (social impact and economic costs related to innovation) (SOL/OP)
   3.2.1 The admin user will be able to manage (CRUD) NEW PARAMETER which must be compatible with the RINF database structure
   3.2.2 Important: this User Story is in “GIS Web Application” and not in “Parameter and formulae of the WEB-APPLICATION” : The admin will be able to input the value of a NEW PARAMETER for a given SOL or OP
   3.2.3 The admin user will be able to add value on SOL/OP selected on the NEW PARAMETER selected
   3.2.4 REST Services : The GIS web application will be able to manage (CRUD) value of New Parameters connected to SOL / OP
   3.2.5 REST Services : The GIS web application will be able to access New Parameter (name and .... Like if its SOL or OP) connected to SOL / OP
   3.2.6 REST Services : The GIS web application will be able to have permission to use R EST Services

3.3 The admin user and application will be able to manage, access and use FORMULAE based on value of parameter (RINF Parameter and New Parameter)
   3.3.1 The admin user will be able to manage FORMULAE connected to PARAMETERS and/or NEW PARAMETERS
      - The admin user will be able to input a formula
      - The admin user will be able to select a parameter (new parameter or RINF parameter)
      - The admin user will be able to create a formula based on selected parameters
      - Permitted Operations: sqr, ^, +, -, *, /, log, ...
      - The admin user can use predefined function (sum of vector, ..)
      - The formula will be applied in the GIS web application for a set of selected SOL
      - The admin user can choose a color set associated with result ranges (of the formula) in order to represent on the map the result of the formula by colorizing SOL/OP
   3.3.2 REST Services : The GIS web application will be able to view, modify and delete FORMULAE
   3.3.3 REST Services : The GIS web application will be able to apply a FORMULA to a SOL set
   3.3.4 REST Services : The GIS web application will be able to have permission to use REST Services
4 GIS WEB APPLICATION

4.2 The user must be able to visualize a map of the railway network on a web application

4.2.1 The user will be able to visualize the railway network
4.2.2 The user will be able to visualize the base map of Open Street Maps (geographical overview)
4.2.3 The user will have an optimal working scale at 1:20 000 000 in the WGS84 coordinate-system.
4.2.4 The user will be able to zoom and move on the map with the mouse
4.2.5 The user will be able to select different layers, e.g. rail network at macro and micro level, population at NUTS level; land use; economic information at NUTS level, station location and characteristics, building at city level, etc.:
   - 1, 2 human and industrial geography (ESR, Eurostat)
   - 3 line characteristics, station and terminal locations (based on Open Street maps)
4.2.6 The user will be able to view a legend on the maps.

4.3 The user must be able to log in, navigate on the map

4.3.1 The user will be able to log in
4.3.2 The user will be able to create/modify/find a login and a password
4.3.3 The user will be able to visualize the SOL (section of line) and the OP (operational point like stations, junctions etc.) of RINF
4.3.4 The user will be able to visualize on the map a layer showing the climate of the different regions
4.3.5 The user will be able to select a language of interface (EN, FR, DE)

4.4 The logged user will be able to select and save a set of RINF SOL/OP in the tab "selection"

4.4.1 The user will be able to reload the last selection performed
   - Optionally, load any stored selection
4.4.2 The user will be able to filter the SOL/OP according to criteria (network/region/technical parameter from RINF/NEW PARAMETER)
   - The user will be able to, first filter the SOL/OP with the following criteria: country
   - The user will be able to visualize the SOL/OP of the network selected on the map (redisplay to the maps) (optional)
   - The user will be able to, second filter the SOL/OP with the following criteria: technical parameters of RINF
   - The user will be able to visualize the SOL/OP filtered by the parameters on the map (redisplay to the maps) (optional)
   - Quantitative or ordered parameters to be identified in RINF
4.4.3 The user will be able to filter SOL/OP by area selection (zoom) on the map, by means of screen limit (boundary)
4.4.4 The user will be able to visualize a list of the filtered SOL/OP by network, region and technical parameters in a geographical zoomed area in the widget/tab maps.
4.4.5 The user will be able to select on the SOL/OP that he/she is interested in
   - The user will be able to select on a list of the SOL/OP that he/she is interested
   - The user will be able to visualize directly (while selecting) the selected SOL/OP items with a distinct colour on the map.
   - The user will be able to name each selected SOL/OP items from the list (optional)
4.4.6 The user will be able to save her/his selection and associate them with a name so that they can access it again later.
4.4.7 The user will have an optimized "user interface" to work with up to 20 selected points of interest
4.5 The admin user will be able to add value for NEW PARAMETERS SOL/OP in the tab "selection"

4.5.1 The admin user will be able to filter the SOL/OP according to criteria (network/region/technical parameter) (4.4.2)
4.5.2 The admin user will be able to filter SOL/OP by area selection (zoom) on the map, by means of screen limit (boundary) (4.4.3)
4.5.3 The admin user will be able to visualize a list of the filtered SOL/OP by network, region and technical parameter in a geographical zoomed area on the screen. (4.4.4)
4.5.4 The admin user will be able to select on the SOL/OP that he/she is interested in (4.4.5)
4.5.5 The admin user will be able to select NEW PARAMETERS for adding values on the SOL/OP selected
4.5.6 The admin user will be able to add value on SOL/OP selected on the NEW PARAMETER selected

4.6 The logged user will be able to view information on a set of SOL/OP (already selected), in the tab "Parameter information/value/..." and on the map, in order to make a decision

4.6.1 The user must be able to select a set of SOL/OP (already selected) to display and export information (note: no link with selection for filtering).
4.6.2 The user must be able to select one or more parameters (from RINF Parameters or NEW PARAMETERS) to display and export information (note: no link with selection for filtering).
4.6.3 The user must be able to visualize the result of the "calculation" (using different methods for extensive vs. intensive values) for all the selected SOL/OPs for each parameter to be explored in the widget/tab.
4.6.4 The user must be able to visualize the result of the "calculation" (kind of count on one parameters of a set of SOL) (extensive and intensive values) for the selected SOL/OP with a pop-up by clicking on the item on the map.
4.6.5 The user must be able to visualize the result of the "calculation" (calculation base on formula using selected parameters) for all the selected SOL to be explored in the widget/tab.
4.6.6 The user must be able to visualize the result of the "formula" (calculation base on formula on some parameters) for the selected SOL/OP with a pop-up by clicking on the item on the map. Each SOL/OP will be colored according to the result value of the formula
4.6.7 The user must be able to export:
   - Set of selected SOL/OP,
   - the parameters values for each SOL/OP,
   - the formula result for each SOL/OP of the set,
   - and the calculation of parameters value for the set of SOL/OP in CSV format (optional PDF, GEOjson)
4.6.8 The user must have an optimized "user interface" to work with up to 4 different parameters

5 Put the system in production
5.1 Deployment of system
5.2 UIC IT to contract a virtual server and install application server and database server
5.3 Update the based data (all geographic information)
2 RINF data

2.1 Presentation of the RINF Data

RINF refers to Article 49 of Directive (EU) 2016/797 and provides for transparency concerning the main features of the European Railway infrastructure. RINF data set is from 2017 and covers 21-member states of the European Union.

The characteristics of the railway infrastructure are described in RINF to unify the information from the different Member States. Although RINF provides infrastructure data at the macro and micro level of the rail network, data at the micro level is not yet available. For this reason, the web-based application makes RINF infrastructure data available at the macro level of the rail network\(^1\).

Some of these definitions of infrastructure features have also adopted by web-based GIS application. Listed below the main infrastructure features of RINF; Line, Section of Line and Operational Point.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{section_of_line.png}
\caption{Section of Line}
\end{figure}

**Line:** A line is a continuous chain of sections of lines and operational points when except beginning and end of a line, the OP at end of a SOL is the OP at start of consecutive SOL. So, a “line” is an aggregation of continuous interconnected OPs and SOLs in such a way that it can be seen as an area-entity with (possibly) common equal characteristics.

\(^{1}\) For more information about micro and macro level: RINF Application guide version 1.2.1
Section of line (SOL): SOL means part of line between adjacent operational points and may consist of several tracks. The SOLs are linked to lines. A single SOL may be settled to be a line at its own. A section of line is the connection between two adjacent OPs.

Operational point (OP): An operational point (OP) is a demarcated area on the micro topology of the railway network which has, as one coherent unit, a function in routing and time-tabling of train services on macro-level. OP also means any location at boundaries between Member States or infrastructure managers.

Even though RINF data set covers 21 Member States, the data is not totally homogeneous for every country. Here is some analysis to view the status of RINF dataset for each of Member States.

Total length of railway network by Member States according to RINF data set:

According to RINF dataset Germany has the longest railway network (32700 km) among the countries that contributed data. France comes in second with 28612 km network, followed by Poland (20000 km), Italy and the United Kingdom (about 15 000 km).
Total number of Line by member states:

Germany is the member state with the highest number of recorded lines in the RINF dataset, with 1270 lines in total. The United Kingdom has more than 1 000 Line in the RINF data set. France comes third on the list with 690 lines records, followed by Poland with 624 lines. There are 6 members states that have less than 30 lines records in the RINF data set.

Average length of Line by Member States:

According to RINF dataset, Sweden has the highest average length of Line with 175 km. Croatia and Bulgaria come both after Sweden with 150 km of average length of Line. Then comes Finland and Latvia with 75 km average length of Line.
**Total number of Section of Line by Member States:**

France is member state with the highest occurrence of Section of Line with 17000 Section of Line records. Runner-Up is Germany (with 8000 SOL), followed by the United Kingdom (5000 SOL) and Poland (4800 SOL). On the other end of the ranking, Greece has only 1 Section of Line record in all its RINF dataset.

**Average length of Section of Line by Member States:**

According to RINF dataset France and Czech Republic have the shortest Section of Line length average (less than 2.5 km). On the other hand, Latvia has the most longest. Section of Line average length (37 km). That’s twice as much as Lithuania, that has an average of only 16km.
We can observe on the map and the bar chart the highly heterogenous and uneven distribution of operational point types across member countries. France comes in first for the total number of operational points, which are mostly technical operational points. Germany is the second country on the operational point record count ranking, but with the highest record of passenger stations. Poland and Czech Republic both have a highly dense record of OPs compared to their land area.
2.2 Set of Parameters exported from RINF

RINF data set contains 98 technical parameters in total and most used 33 parameters for Section of Line and Operational Points are implemented in the GIS Web Based Application. List of 33 selected RINF parameters:

**Infrastructure Subsystem**

*Performance parameters*

- **Part of a Railway freight corridor (IPP_FreightCorridor):** Indication whether the line is designated to a Railway Freight Corridor
- **Maximum permitted speed (IPP_MaxSpeed):** Nominal maximum operational speed on the line
- **Maximum altitude (IPP_MaxAltitude):** Highest point of the section of line above sea level
- **Temperature range (IPP_TempRange):** Temperature range for unrestricted access to the line according European standard.
- **Category of line (IPP_LineCat):** Classification of a line according to the INF TSI.
- **TEN classifications of track (IPP_TENClass):** Indication of the part of the trans-European network the line belongs to.

*Line layout*

- **Standard combined transport profile number for swap bodies (ILL_ProfileNumSwapBodies):** Coding for combined transport with swap bodies as defined in UIC Code.
- **Gradient profile (ILL_GradProfile):** Sequence of gradient values and locations of change in gradient
- **Minimal radius of horizontal curve (ILL_MinRadHorzCurve):** Radius of the smallest horizontal curve of the track in metres

*Track parameters*

- **Cant deficiency (ITP_CantDeficiency):** Cant deficiency is present when a vehicle's speed on a curve is greater than the speed at which the components of wheel to rail force are normal to the plane of the track. In RINF, maximum cant deficiency expressed in millimetres defined as difference between the applied cant and a higher equilibrium cant the line has been designed for.
- **Rail inclination (ITP_RailInclination):** An angle defining the inclination of the head of a rail relative to the running surface.
- **Existence of ballast (ITP_Ballast):** Specifies whether track construction is with sleepers embedded in ballast or not.
Energy System

**Contact line system**

- **Type of contact line system (ECS_SystemType):** Indication of the type of the contact line system.
- **Maximum train current (ECS_MaxTrainCurrent):** Indication of the maximum allowable train current expressed in amperes.
- **Maximum contact wire height (ECS_MaxWireHeight):** Indication of the maximum contact wire height expressed in metres.
- **Minimum contact wire height (ECS_MinWireHeight):** Indication of the minimum contact wire height expressed in metres.
- **Permission for regenerative braking (ECS_RegenerativeBraking):** Indication whether regenerative braking is permitted or not.

**Pantograph**

- **Accepted TSI compliant pantograph heads (EPA_TSIHeads):** Indication of which TSI compliant pantograph heads are allowed to be used.
- **Accepted other pantograph heads (EPA_OtherHeads):** Indication of which pantograph heads are allowed to be used
- **Permitted contact strip material (EPA_StripMaterial):** Indication of which contact strip materials are permitted to be used.

Control-Command and Signalling Subsystem

**TSI compliant train protection system (ETCS)**

- **ETCS level (CPE_Level):** ERTMS / ETCS application level related to the track side equipment.
- **ETCS baseline (CPE_Baseline):** ETCS baseline installed lineside.

**Train protection legacy systems**

- **Existence of other train protection, control and warning systems installed (CPO_Installed):** Indication if other train protection, control and warning systems in normal operation are installed lineside.

**Train detection systems not fully compliant with the TSI**

- **Minimum permitted width of the rim (CTD_MinRimWidth):** Indication of width given in millimetres.
Minimum permitted wheel diameter (CTD_MinWheelDiameter): Indication of wheel diameter given in millimetres.

Minimum permitted thickness of the flange (CTD_MinFlangeThickness): Indication of flange thickness given in millimetres.

Maximum permitted height of the flange (CTD_MinFlangeHeight): Indication of height of flange given in millimetres.

Minimum permitted axle load (CTD_MinAxleLoad): Indication of load given in tons.

Platform and Tunnel

**Platform**

**TEN Classification of platform (IPL_TENClass):** Indicates the part of the trans-European network the platform belongs to. (Comprehensive Network, Core Freight Network, Core Passenger Network Off-TEN)

**Existence of platform assistance for starting train (IPL_AssistanceStartingTrain):** Indication of existence of equipment or staff supporting the train crew in starting the train.

**Tunnel**

**Existence of emergency plan (ITU_EmergencyPlan):** Indication (Yes or No) whether emergency plan exists.

**Fire category of rolling stock required (ITU_FireCatReq):** Categorisation on how a passenger train with a fire on board will continue to operate for a defined time period. (A, B, None)

**National fire category of rolling stock required (ITU_NatFireCatReq):** Categorisation on how a passenger train with a fire on board will continue to operate for a defined time period - according to national rules if they exist.

### 2.3 Conclusion

Standardised and publicly available railway infrastructure data covering European railway infrastructure are only included in the Infrastructure Register which has been prepared by the European Railway Agency. For this reason, the application is grounded in RINF data for rail network infrastructure information. Also, the application will likely provide and update infrastructure data in RINF data set formats.
3 Benefits calculation and decision-making aids

3.1 Fundamental option: treat code like data

This is maybe the most unusual, if not innovative, part of the web application.

Estimating the benefits of innovations rests on functions applied to sets of sections of lines. For instance, introducing trolley wires may reduce costs on electrified lines, but obviously not elsewhere; the corresponding function is expected to take the SoL electrification parameters into account and return a likely cost saving, resting on the CBA or through other means.

Rather than hard-coding such functions, we decided to consider them as advanced user input. The function is then provided in textual form, respecting the syntax of the Python programming language.

This option brings about specific security issues, since it offers the theoretical possibility of injecting malicious code that may affect the server. However the risk is low, since for the time being, function input is reserved to registered users, and the input does not allow all syntactic constructs that the Python language would accept. In the very near future, UIC intends to increase security by parsing the functions submitted by the user and forbid the access to certain libraries.

Description of useful functions is provided under D6.5 and D6.6 (using common mathematics notation).
4 Infrastructure specification for host web-application

4.1 Technical Architecture Document

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

From an organizational point of view, the system is structured into three sub-systems (see Figure 4.1) which interacts dynamically with each other, and which have distinct functional responsibilities.

![Figure 4.1 the Organizational structure of the system](image)

GIS Data Base has as main responsibilities the storage of system data as well as the execution of database queries.

Rest API is running in the background and has as main responsibility the provision of information in standard GIS format (GeoJson2).

GIS Web Application provides the interface between the users and the database of the system, and last but not least interaction with the Rest API in order to provide information in a graphic format.

---

2 GeoJson - is a 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.
4.1.1 Global architecture scheme

From a conceptual point of view, we designed an architecture structured into tree modules, namely (see Figure 4.2):

- User Interface
- Application Server
- Data Store

![Figure 4.2 Global system architecture](image)

Designing this architecture took into account the specific business requirements (described in section 1.2), and also gives possibility to easily combine individual components in an existing IT infrastructure.

- **User Interface**

For a user interface that operate equally well on handheld devices and on desktop browsers, we used modern front-end frameworks such as **JQuery** and **Bootstrap** together with a mapping component – **Leaflet**.

- **Leaflet** - right now, is the most popular javascript map library, with many available plugins.
- **Bootstrap** - is an open source toolkit for developing with HTML, CSS, and JS. Bootstrap also provides 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, by means of an easy-to-use API that works across a multitude of browsers.
For desktop *data preparation*, 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 needed.

- **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 settle business logic outside of the Geographic DB Server (e.g. PostgreSQL with PostGIS extension) and develop our own web application framework based on the Django Framework.

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

In our application server, we use 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 Rest Framework** - is a powerful and flexible toolkit for building Web Rest APIs (1)
- **Django Rest Framework GIS** – is an opensource Geographic add-ons for Django Rest Framework (2)
- **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 databases than in NoSQL databases. In our project, we preferred the more mature relational DB approach but would consider NoSQL in cases where data volume could grow unexpectedly very large, and 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 when preparing the data.
- **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, hyper logs 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.
4.1.2 Technical platform
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 using the Django framework. Specifically, this framework allowed us to build a system based on a standard MTV (Model-Template-View, see Figure 4.3) web architecture:

- **Model** - the data access layer. This layer contains anything and everything about the data: how to access it, how to validate it, which behaviour 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.

In order to achieve our task goals in manipulating geographic data, we had to integrate more software packages into Django framework (see Figure 4.4), namely:

- **Django psycopg2** – is a PostgreSQL + PostGIS adapter for Python
- **Geo Django** - is a 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** - 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 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

![Figure 4.4 NetIRail Web GIS architecture based on Django Framework](image-url)
4.2 Class diagrams
According to the levels described in the previous chapter, specific classes were implemented for each of the system physical levels.

4.2.1 GIS class diagrams
This layer of our Web GIS system works at the server-side level, and is implemented on a REST API architecture.

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 4.5)
For providing operations to handle geographic data and associated metadata (CRUD: Create, Read, Update and Delete)

Figure 4.6 Class Diagram for operations to handle geographic data and associated metadata

For creating query data based on dynamic filters

Figure 4.7 Class diagram for creating query data based on dynamic filters
1.1.1 RINF class diagram
For providing operations to handle RINF data and associated metadata (Create, Read, Update and Delete)

Figure 4.8 Class Diagram for operations to handle RINF data and associated metadata
4.2.2 Client-side application class

This class 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.

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 4.2.1.

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 4.9 and Figure 4.10).
To manipulate interface elements in an AJAX manner, we have integrated the JQuery library (4).

The most important integrated library in our application is Leaflet (5). This javascript library enables us to build dynamic maps as well as implement controls for the selection and display of geographic layers, to implement localization, zooming and map dragging functions (see Figure 4.12).
Figure 4.12 Integration diagram of Leaflet library in our GIS application

Figure 4.13 Simplified class diagram of Leaflet used in our GIS application (6)
4.3 Data Description

4.3.1 Relational Database Structure
The database for the decision support tools consists of the following 2 types of tables:

- Tables dedicated to GIS components
- Table dedicated to RINF components

4.3.1.1 Tables dedicated to GIS components
The database dedicated to GIS components consists of the following tables (see Figure 4.14):

- **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) and also for management of the geographic representation density of population at NUTS level
- **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 (e.g., 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.

There are 4 types of geographic elements for each layer, namely:

- Point,
- LineString,
- Polygon,
- MultiPolygon.

Spatial reference system for all types of geometries is SRID 4326, and for graphic representation, WSG84 is used.
Figure 4.14 Database structure for GIS components
4.3.1.2 Tables dedicated to RINF components

The database dedicated to RINF components consists of the following tables:

- **Organizations** - table responsible for the management of organizations of users
- **Documents** - table responsible for the management of RINF documents imported in the system
- **RINF_memberstates** - table responsible for the management of RINF Member States
- **RINF_lines** - table responsible for the management of RINF Lines
- **RINF_Sol** - table responsible for the management of RINF Sections of Lines
- **RINF_Op** - table responsible for the management of RINF Operational Points
- **RINF_Par** - table responsible for the management of RINF Parameters
- **RINF_Sol_Par_Val** - table responsible for the management of RINF Values of Parameters
- **RINF_Op_Type** - table responsible for the management of RINF Types of Operational Points
- **RINF_Op_Val** - table responsible for the management of RINF Values of Parameters for Operational Points
- **RINF_Par_Def** - table responsible for the management of RINF Definitions of Parameters
- **RINF_Formula** - table responsible for the management of formula for calculation score for different characteristics of Sections of Lines
- **RINF_Formula_Parameters** - table responsible for the management of Parameters used in a specific Formula
- **RINF_Sol_Results_of_Formula** - table responsible for the management of Results of calculation for specific Formula at level of Sections of Lines

There are 2 types of geographic elements for Table RINF_Op and RINF_Sol, namely:

- Point
- MultiLineString

Spatial reference system for all types of geometries is SRID 4326, and for graphic representation, WSG84 is used.
Figure 4.15 Database structure for RINF components
4.3.2 **Physical Model**

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 behaviors of the data we store. In general, each class in the model correspond to a single table of the database. That's why our implementation model in Python has the same structure as our database.

![Figure 4.16 Physical Model of GIS Rest API](image)
Figure 4.17 Physical Model of RINF Rest API
4.4 Code Description

4.4.1 Code architecture

In this section, the approach to apply “Clean” Architecture (see Figure 4.18) on a Django Restful API is illustrated.

Various approaches to code architecture are defined in the IT literature, for example:

- Hexagonal Architecture (8) (a.k.a. Ports and Adapters) by Steve Freeman, and Nat Pryce
- Onion Architecture by Jeffrey Palermo (9)
- DCI from James Coplien, and Trygve Reenskaug (10)
- BCE by Ivar Jacobson (11)

Each of these architectures produces systems that are:

- Independent of Frameworks. Architecture does not depend on the existence of a particular software library with loaded features. This will allow the use of such framework as tools.
- Testable. Business rules can be tested without the user interface, database, Web server, or any other external feature.
- Independent of the user interface. The user interface can easily change without changing the rest of the system.
- Independent database. The architecture offers the possibility to change the database without changing the Business Rules.

The Clean Architecture is integrating all these architectures into a single architecture.

Our code architecture is structured by the same layers as the diagram above (see Figure 4.18).

- Entities Layer (Innermost Domain)

Below is a simplistic example of an entity. A real entity must be richer. A real entity assigns business logic and high-level rules that are related to this entity (e.g., Invariant Validations).
Use Cases Layer (Outermost Domain)

The use cases are called eg. interactors.py and they contain the business logic of each use case. And here all the application logic is placed. A command pattern for their implementation is used because it helps with task enqueueing, rollback when an error occurs and also separates dependencies and parameters (really useful for readability, testing and dependency injection).

Interface Adapters Layer

In this layer are the pieces that are decoupled from framework, but are conscious of the environment (API Restful, database storage, caching, etc.).

First of all, there is views.py. They follow Django’s view structure but are completely decoupled from it.
4.4.2 Identification and authentication mechanisms

Django Framework (12) comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The authentication system consists of:

- Users
- Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
- Groups: A generic way of applying labels and permissions to more than one user.
- A configurable password hashing system
- Forms and view tools for logging in users, or restricting content
- A pluggable backend system

The authentication system in Django aims to be very generic and doesn’t provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

- Password strength checking
- Throttling of login attempts
4.4.3 Importation of RINF data: mechanism

The import of data is done in the background of the application and involves the execution of the following processes:

- upload the RINF xml file into the system
- extract data from the imported xml file
- validating and transforming data extracted in the format of our ORM model
- Inserting validated and converted data into the system database

Because RINF files for a Member State are of considerable size (e.g. for France, size exceeds 500Mb), we had to use the chunk technology. This technology allows us to break a large file into small chunks and send these pieces to the upload server one by one. If an upload fails, we need only resume from the last successful chunk.

This also gives access to another important concept: pausing and resuming uploads. In the case of uploading multiple files at once, this allows us to de-prioritize non-essential files.
The process for extracting data, validating and transforming data extracted and also inserting validated/converted data into the system database works exclusively in background transparent to the user (see Figure 4.23).

**Figure 4.23** Class diagram for extracting data, validating and transforming data extracted and also inserting validated/converted data into the system database
4.4.4 Architecture of the formula mechanism

One of the main innovations brought to our system is the implementation of a mechanism for assigning the dynamics of some formulas that use the SOL parameters.

Thus, the user can select parameters they want to use in a formula, then assign them a "nickname" (e.g. P1 for the IPP_TENClass-10-Part of the Ten-T Comprehensive Network) because the full name of this is difficult to use in formulas. And after selecting these parameters, the user can edit the mathematical formula using the nickname of these parameters (see Figure 4.25).

Editing the mathematical formula is done in the latex style, eg “\text{sum}(P1)+1.3\log(P2\times P3)*\exp(\sqrt{P3})”

We have to mention that in our system we use 2 types of parameters:

- Parameters imported from the RINF database (default parameters)
- User-created parameters (custom parameters)

For default parameters, data is automatically assigned to RINF files, for custom parameters the data must be manually assigned by the user for each SOL.

It is important that within a formula the user can use both default parameters and custom parameters.

![Figure 4.24 Formula management form](image-url)
Moreover, in order to use the parameter string values of the parameters within the formulas, a set of numerical transformation functions have been implemented in the created system. This allows to replace descriptive parameters, such as the electrification type, into a set of dummy variables taking the value 1 (corresponding to the chosen type of electrification) or 0 (corresponding to other types); see Deliverable D6.5 for more information about dummy variables.

After applying the formulas to the data parameters of a SoL, their result can be consulted by the user at each SoL level within the GIS application.

4.4.5 ORM
ORM is the acronym for Object Relational Mapping. There are many implementations of ORM on different platforms, frameworks, or programming languages. In fact, an ORM framework is wrapped around a relational database. The object classes are mapped to the data tables in the database, and the object instances are mapped to the rows in those tables.

The following are advantages of ORM:

- **Database Independent** - No need to write code specific to a particular database. Changing a few lines of code in the database configuration adapter makes it work with another database.
- **Domain Model Pattern** – ORM lets the business code access objects rather than the database structure itself, using the mapping between the business model and the database.
- **Reduce Code and Increase Efficiency** - ORM provides an abstraction which allows the developers to focus on their business logic, rather than complex database queries, resulting in a huge reduction of code and increase in efficiency of the developer.
- **Relationships** - ORM provides a hassle-free management of database relationships. The related objects are automatically loaded when a query is translated to its corresponding SQL.
- **Concurrency** - ORM supports concurrency, allowing multiple users to update the same set of data simultaneously.
- **Caching** - Objects can be cached in the memory, reducing the load on the database.
- **Transactions** - The changes to an object can be bound to a transaction, which can either be committed or rolled back. Multiple transactions can be active at a given point of time, but all these transactions are isolated from one another.

In our system the ORM implementation of Django frameworks is used, and Django rest framework.

![Figure 4.26 Example of ORM implementation in Django, Parameter class](image)

### 4.4.6 Leaflet

In the NeTIRail-INFRA system, leaflet libraries play an important role, because they represent maps, and last but not least, they offer layers handling functions, as well as providing management capabilities for maps (zoom, location, movement, etc.).

Leaflet.js is an open-source library using which we can deploy simple, interactive, lightweight web maps.

- Leaflet JavaScript library allows you to use layers such as Tile layers, WMS, Markers, Popups, Vector layers (polylines, polygons, circles, etc.), Image overlays and GeoJSON.
- Using Leaflet JavaScript, we can interact with the Leaflet maps by dragging the map, zooming (by double click or, wheel scroll), using keyboard, using event handling, and by dragging the markers.
- Leaflet supports browsers such as Chrome, Firefox, Safari 5+, Opera 12+, IE 7–11 on desktops, and browsers like Safari, Android, Chrome, Firefox for mobiles.

The simplified class diagram of Leaflet used in our GIS application was presented in Figure 4.13., and we will continue to present the most important classes (L.map, L.control etc) used by us in the project.

In the following table shows the name of the event, the object returned when the event fires, and a description for Map Control Events (L.map events).
<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>When Fired</th>
</tr>
</thead>
<tbody>
<tr>
<td>click</td>
<td>MouseEvent</td>
<td>When the user clicks (or taps) the map</td>
</tr>
<tr>
<td>dblclick</td>
<td>MouseEvent</td>
<td>When the user double-clicks (or double-taps) the map.</td>
</tr>
<tr>
<td>mousedown</td>
<td>MouseEvent</td>
<td>When the user pushes the mouse button while over the map.</td>
</tr>
<tr>
<td>mouseup</td>
<td>MouseEvent</td>
<td>When the user releases the mouse button while over the map.</td>
</tr>
<tr>
<td>mouseover</td>
<td>MouseEvent</td>
<td>When the mouse pointer enters the map</td>
</tr>
<tr>
<td>mouseout</td>
<td>MouseEvent</td>
<td>When the mouse pointer leaves the map</td>
</tr>
<tr>
<td>mousemove</td>
<td>MouseEvent</td>
<td>While the mouse pointer is moving over the map</td>
</tr>
<tr>
<td>contextmenu</td>
<td>MouseEvent</td>
<td>When the user pushes the right mouse button while over the map (or uses a long press on a mobile device). If you handle this event, it prevents the usual context menu from appearing.</td>
</tr>
<tr>
<td>focus</td>
<td>Event</td>
<td>When the map control gains the focus (via clicking, panning, or tabbing to it).</td>
</tr>
<tr>
<td>load</td>
<td>Event</td>
<td>When the map is initialized and its center and zoom level is fixed for the first time.</td>
</tr>
<tr>
<td>movestart</td>
<td>Event</td>
<td>When the map extent starts to change (such as when the user starts dragging the map).</td>
</tr>
<tr>
<td>moveend</td>
<td>Event</td>
<td>When the map extent stops changing (such as when the user stops dragging the map).</td>
</tr>
<tr>
<td>zoomstart</td>
<td>Event</td>
<td>When the map zoom level is about to change (before zoom animation occurs).</td>
</tr>
<tr>
<td>zoomend</td>
<td>Event</td>
<td>When the map zoom level changes.</td>
</tr>
<tr>
<td>zoomlevelschange</td>
<td>Event</td>
<td>When the number of zoom levels in the map changes due to the addition or removal of a layer.</td>
</tr>
<tr>
<td>layeradd</td>
<td>LayerEvent</td>
<td>When a new layer is added to the map</td>
</tr>
<tr>
<td>layerremove</td>
<td>LayerEvent</td>
<td>When a layer is removed from the map</td>
</tr>
<tr>
<td>baselayerchange</td>
<td>LayerEvent</td>
<td>When the layer control changes the current base layer.</td>
</tr>
<tr>
<td>overlayadd</td>
<td>LayerEvent</td>
<td>When the layer control selects an overlay.</td>
</tr>
<tr>
<td>overlayremove</td>
<td>LayerEvent</td>
<td>When the layer control deselects an overlay.</td>
</tr>
</tbody>
</table>

**Table 4.1 Map Control Events**

In order to work with events, we need to know:

- Which event you want your application to start noticing
- The name of the event
- The type of data returned by the event when it fires
Regarding the geographical elements, these are divided into two categories namely:

- Base maps provide a “background” for the data that forms the basis of your application, which is typically in the form of an overlay.
- Layer data - application data are in the form of points, lines, or polygons (see section 4.3.1.1). This data is rendered dynamically on our map at runtime. Leaflet.js provides a number of different classes to help us do this.

```javascript
var rail_station = L.polyline([]),

pointLayer: function (feature, latlng) {
    return L.marker(latlng,
        {icon: L.icon({
        iconUrl: "static/img/station.png"
        })
    })
},

onEachFeature: function (feature, layer) {
    var _name = "unknown station";
    if (feature.properties.name !== null) {
        _name = feature.properties.name;
    }
    layer.bindPopup("<pre>" + JSON.stringify(feature.properties, null, ' ').replace(/\[\]/g, '').replace(/\"/g, '') + '</pre>");
},

$.getJSON(RailStationURL, function (data) {
    setTimeout(function () {
        rail_station.addData(data);
        map.spin(false);
    }, 200);
});
```

Figure 4.28 Rendered dynamically data on map(a), Load data on map(b)
4.4.7 GIS
A geographic information system (GIS) is a framework for gathering, managing, and analyzing data.

In fact, in our system we were implemented the following components of a GIS framework:

- **Maps** - are the geographic container for the data layers and analytics you want to work with. GIS maps are easily shared and embedded in apps, and accessible by virtually everyone, everywhere via web interfaces.
- **Data** - GIS integrates many different kinds of data layers using spatial location. GIS data includes features, and base maps linked to specific database tables (see section 4.3.1.1).
- **Analysis** - Spatial analysis lets us evaluate suitability and capability, estimate and predict, interpret and understand, and much more, lending new perspectives to your insight and decision-making.
- **Apps** - Apps provide focused user experiences for getting work done and bringing GIS to everyone. GIS apps can work virtually everywhere: in web browsers, and on desktops.

In the next figure are presented the principle features of our GIS framework.

![The principle features of our GIS framework](image)

**Figure 4.29** The principle features of our GIS framework

As basic data, the NeTIRail-INFRA system manages a multitude of specific layers that are stored in the database. The most important are rail infrastructure; they are imported from the open street map in format pbf.

The OSM data are structure in tree category of data, namely nodes, ways and relations:

- **Nodes** – A node represents a specific point on the earth's surface defined by its latitude and longitude. Each node comprises at least an id number and a pair of coordinates. Nodes can be used to define standalone point features.
- **Ways** – A way is an ordered list of between 2 and 2,000 nodes that define a polyline. Ways are used to represent linear features such as rail network.
- **Relations** – A relation is a multi-purpose data structure that documents a relationship between two or more data elements (nodes, ways, and/or other relations).

In order to extract the data from the OSM pbf file within our system, a mechanism has been created based on 2 open-source libraries, namely: osm2pgsql and osmconvert. Through this mechanism the

---

3 Protocol Buffer Format - is a compact binary format that is smaller to download and much faster to process and should be used when possible
4 osm2pgsql - is a tool for loading OpenStreetMap data into a PostgreSQL / PostGIS database suitable for applications like rendering into a map, geocoding or general analysis.
5 osmconvert - is a tool for to convert and process OpenStreetMap files.
app only extracts data relevant to railway infrastructure, which can then be introduced into the specific tables.

Since the extracted geographic data is not routed, a mechanism has been created based on `pgr_createTopology` and `pgr_dijkstra` in order to obtain a routable rail network. Because in the files in the RINF file structure, SOL are defined only by the start and end points, by means implementing the routing function, we can get the geometry of entire sections in the form of a line.

Also, the data imported in previous steps must be transformed to fit our database structure. Transformation is done for both data imported from OSM files and from RINF files.

For OSM files, transformation applies to metadata that are stored in hstore format and must be extracted in specific fields.

For RINF files, transformation applies to latitude and longitude data for point definition in OP and SOL in order to transform it into geometry fields.

---

6 `pgr_createTopology` – function to builds a network topology based on the geometry information.

7 `pgr_dijkstra` - Returns the shortest path(s) using Dijkstra algorithm. Dijkstra’s algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956. It is a graph search algorithm that solves the shortest path problem for a graph with non-negative edge path costs, producing a shortest path from a starting vertex (start_vid) to an ending vertex (end_vid). This implementation can be used with a directed graph and an undirected graph.

8 `hstore` - HStore is a key value store within Postgres. The text representation of an hstore, used for input and output, includes zero or more key => value pairs separated by commas.
For the geographic data management within our system, we have implemented a GIS rest API application, through which you can query the geographic database as well as CRUD operations. This API interacts with both basic data (OSM data) and RINF data.

![List of managed geographic layer](image)

**Figure 4.32** List of managed geographic layer

![Form for management data for a specific geometry](image)

**Figure 4.33** Form for management data for a specific geometry

### 4.4.8 Django REST

In the NeTIRail-INFRA system the following REST components were implemented: Django Rest Framework and Django Rest Framework GIS.

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. All in a fully REST compliant wrapper.

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also
provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

The serializers in REST framework work very similarly to Django's Form and ModelForm classes. We provide a Serializer class which gives you a powerful, generic way to control the output of your responses, as well as a ModelSerializer class which provides a useful shortcut for creating serializers that deal with model instances and querysets.

REST framework also allows you to work with regular function-based views. It provides a set of simple decorators that wrap your function-based views to ensure they receive an instance of Request (rather than the usual Django HttpRequest) and allows them to return a Response (instead of a Django HttpResponse), and allow you to configure how the request is processed.

The Django Rest Framework GIS is add-ons for Django Rest Framework dedicated to GIS functionality.

The Django Rest Framework GIS extend Django Rest Framework with following features:

- Provides a GeometryField, which is a subclass of Django Rest Framework (from now on DRF) WritableField. This field handles GeoDjango geometry fields, providing custom to_native and from_native methods for GeoJSON input/output.
- Provides a GeoModelSerializer, which is a subclass of DRF ModelSerializer. This serializer updates the field_mapping dictionary to include field mapping of GeoDjango geometry fields to the above GeometryField.
- Provide a GeometryFilter field as well as a GeoFilterSet for usage with django_filter. You simply provide, in the query string, one of the textual types supported by GEOSGeometry. By default, this includes WKT, HEXEWKB, WKB (in a buffer), and GeoJSON.
- Provides a InBBoxFilter, which is a subclass of DRF BaseFilterBackend. Filters a queryset to only those instances within a certain bounding box.

Based on these libraries in our system, we have implemented rest modules for geographic resource management.

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

NeTIRail GIS- Rest API provides the basic features of a GIS application by providing user functions for

- displaying geographic (rail network, rail stations, land use, population densities, places etc.) in GeoJson format
- filtering layers
- display layer metadata in GeoJson format
- zoom in / zoom out functions

Moreover, through this component, we 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.

Also, at our Rest GIS API we implemented a Resy 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 4.34 The GIS REST API interface for accessing principal functions for the manipulation of GIS data

Figure 4.35 Example of Rest API interface for Get (a) and Post (b) data
4.4.9 Management of multiple languages

The goal of internationalization and localization is to allow a single Web application to offer its content in languages and formats tailored to the audience.

Django Framework offers full support for translation of text, formatting of dates, times and numbers, and time zones.

In order to translate the text of the web application it is necessary to implement i18N (which is the process of planning and implementing products and services so that they can easily be adapted to specific local languages and cultures, a process called localization).

First, we created a folder to save all the translation files. Those files are created automatically by Django, writing the strings that we want to translate.

The *ugettext_lazy* function is used to mark the language names for translation, and it is usual to use the function’s shortcut _.

![Figure 4.36 Example of REST API interface for Put and Delete data](image)

![Figure 4.37 Settings internalization support in our Django application](image)
To mark all the fields that are supposed to be translated into the template file named base.html I have introduced translation tags:

```
{% load i18n %}
```

In each web form, the fields to be translated must be typed into a translation tag of the following type:

```
{% trans %} or {% blocktrans %} .... {% endblocktrans %}
```

Django offers an automatic mechanism for extracting information that is meant to be translated (marked with {% trans %} or {% blocktrans %} .... {% endblocktrans %}), generating a database for each language in format txt.

The command for extracting this information is the following:

```
manage.py@NetIRail-decision-support-tool makemessages -l fr
```

As a result of this command is a text file with the extension of the languages in which transplantation is given (e.g. fr, etc.).

Each of these sentences appear in a line beginning with `msgid`. We have to put your translation in the next line, the one that starts with `msgstr`, e.g.:

```
msgid "Log in to NeTIRail Decision Support Tool!"
msgstr "Connectez-vous à l’outil d’aide à la décision NeTIRail!"
```
4.4.10 Description of the application log process
Django uses Python’s built-in logging module to perform system logging.

A Python logging configuration consists of four parts:
- Loggers
- Handlers

Each message that is written to the logger is a Log Record. Each log record also has a log level indicating the severity of that specific message. A log record can also contain useful metadata that describes the event that is being logged. This can include details such as a stack trace or an error code.

When a message is given to the logger, the log level of the message is compared to the log level of the logger. If the log level of the message meets or exceeds the log level of the logger itself, the message will undergo further processing. If it doesn’t, the message will be ignored.

The handler is the engine that determines what happens to each message in a logger. It describes a particular logging behavior, such as writing a message to the screen, to a file, or to a network socket.

A logger can have multiple handlers, and each handler can have a different log level. In this way, it is possible to provide different forms of notification depending on the importance of a message. For example, you could install one handler that forwards ERROR and CRITICAL messages to a paging service, while a second handler logs all messages (including ERROR and CRITICAL messages) to a file for later analysis.

4.4.11 Description of the ERROR management process
Error management is closely related to the logging process. The recording of the possible is made in the error execution log form.

For each identified error, a severity level is assigned, according to the following table:

<table>
<thead>
<tr>
<th>Severity level</th>
<th>Effect on the 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 4.2 Error severity level

4.4.12 Libraries used
Most libraries used have been described in previous chapters. Here we will just summarize them:
<table>
<thead>
<tr>
<th>Name</th>
<th>Application layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>Data sources</td>
<td>RDBS system</td>
</tr>
<tr>
<td>PostGIS</td>
<td>Data sources</td>
<td>PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.</td>
</tr>
<tr>
<td>PostGIS Topology</td>
<td>Data sources</td>
<td>The PostGIS Topology types and functions are used to manage topological objects such as faces, edges and nodes.</td>
</tr>
<tr>
<td>pgrouting</td>
<td>Data sources</td>
<td>Extends the PostGIS / PostgreSQL geospatial database to provide geospatial routing functionality.</td>
</tr>
<tr>
<td>Redis Server</td>
<td>Data sources</td>
<td>It 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, hyper logs 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.</td>
</tr>
<tr>
<td>Django Framework</td>
<td>Application server</td>
<td>It is an opens source high-level Python Web framework that encourages rapid development and clean, pragmatic design.</td>
</tr>
<tr>
<td>Django Rest Framework</td>
<td>Application server</td>
<td>Provides powerful model serialization, display data using standard function-based views, or get granular with powerful class-based views for more complex functionality.</td>
</tr>
<tr>
<td>Django Rest Framework</td>
<td>Application server</td>
<td>The Django Rest Framework GIS is add-ons for Django Rest Framework dedicated to GIS functionality</td>
</tr>
<tr>
<td>Django-Leaflet</td>
<td>Application server</td>
<td>Allows us to use Leaflet java script library in our Django projects.</td>
</tr>
<tr>
<td>Django-GeoJSON</td>
<td>Application server</td>
<td>is a set of tools to manipulate GeoJSON data with Django:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- (De)Serializer for (Geo)Django objects, querysets and lists</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Base views to serve GeoJSON map layers from models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GeoJSON model and form fields to avoid spatial database backends (compatible with django-leaflet for map widgets)</td>
</tr>
<tr>
<td>Name</td>
<td>Application layer</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Django-Redis</td>
<td>Application server</td>
<td>is a full featured Redis cache/session backend for Django</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>Application server / Client interface</td>
<td>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.</td>
</tr>
<tr>
<td>JQuery</td>
<td>Client interface</td>
<td>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.</td>
</tr>
<tr>
<td>leaflet-sidebar-v2</td>
<td>Leaflet plugin/ Client interface</td>
<td>A responsive sidebar plugin for Leaflet, a JS library for interactive maps.</td>
</tr>
<tr>
<td>leaflet-groupedlayercontrol</td>
<td>Leaflet plugin/ Client interface</td>
<td>Leaflet layer control with support for grouping overlays together. Also supports making groups exclusive (radio instead of checkbox).</td>
</tr>
<tr>
<td>leaflet-locatecontrol</td>
<td>Leaflet plugin/ Client interface</td>
<td>A leaflet control to geolocate the user</td>
</tr>
<tr>
<td>leaflet-markercluster</td>
<td>Leaflet plugin/ Client interface</td>
<td>Provides Animated Marker Clustering functionality for Leaflet, a JS library for interactive maps.</td>
</tr>
<tr>
<td>leaflet-ajax</td>
<td>Leaflet plugin/ Client interface</td>
<td>Allows you to call JSON via an Ajax call with a jsonp fallback.</td>
</tr>
<tr>
<td>leaflet-spin</td>
<td>Leaflet plugin/ Client interface</td>
<td>Shows a spin cursor on the map during process execution</td>
</tr>
<tr>
<td>leaflet-geojson-vt</td>
<td>Leaflet plugin/ Client interface</td>
<td>A highly efficient JavaScript library for slicing GeoJSON data into vector tiles on the fly, primarily designed to enable rendering and interacting with large geospatial datasets on the browser side (without a server).</td>
</tr>
</tbody>
</table>

Table 4.3 Library used in NeTIRail GIS web system
4.5 HARDWARE PLATFORMS

4.5.1 Recommended environment

Hardware requirements: 1 dedicated virtual or physical server

- CPU - 2 x octa-core processors recommended (min 12 core)
- RAM - 32 GB recommended (min 8GB)
- + 700 GB HDD recommended (min 300GB)

Software requirements:

- OS – Linux -Red Hat / Centos / Ubuntu / Debian or Windows 2012 R2
- DB server - PostgreSQL 10.x + PostGIS + pgRouting
- Web server - Apache, nginx or lighttpd
- Cache server – Redis
- Python 3.6.x + Django 2.x

4.5.2 Performance specifications

The server must be able to serve up to 50 concurrent users.

Also, the necessary bandwidth must be min 500Mbit/s in order to ensure capability for download / upload of more then 500MB of data per session / query.
5 Conclusion

The decision support tool is deliberately data- (RINF, OpenStreetMap) and knowledge- (custom evaluation functions) centric. Its feature set has been kept relatively low and can easily be enriched, given the robust software architecture chosen. However, its main strength is easy customization, by means of user-defined data and functions.
6 References


