POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Politecnico
di Torino

Master's Degree Thesis

A web-based georeferenced modelling tool for the Turin tramway network electrification system

Supervisor

Prof. Enrico PONS

Co-supervisors

Prof. Edoardo PATTI

Prof. Lorenzo BOTTACCIOLI

Dott. Pietro COLELLA

Candidate

Francesco BARLOCCO

JULY 2021

Abstract

This thesis, developed as part of a collaboration between Infra.To and Politecnico di Torino, describes the development of a web application for the modelling and simulation of the Turin tramway network electrification system. The destination of the work is two-fold: on the one hand it should allow network operators to perform quantitative analyses of the system, in order to support both maintenance and planning phases by studying hypothetical scenarios. On the other hand, it should contribute to a research project of the Energy Center Lab at Politecnico di Torino that aims to provide a comprehensive monitoring of the energy flows in the city. The tool is structured around a web GIS platform, based on the MapStore framework, which lets display and edit, on a map, the georeferenced data of the network infrastructure, imported from the AutoCAD environment. The application includes a simulator, developed in Python, that builds the steady-state circuital model of the DC traction system and solves it by means of nodal analysis. The simulator can either analyse a selected configuration of trams and faults, or compute the minimum short-circuit current for a given zone of the overhead contact system. Simulations results have been validated through a comparison with those obtained on the LTspice software, showing the correctness of the algorithms that build and solve the circuital model of the network. In general, the criteria of flexibility and openness are the ones that have mainly driven the design choices: all the software components are open-source, interact through RESTful APIs and implement the standard protocols defined by OGC for the sharing of geospatial data through the web. This should facilitate possible future expansions of the work: for instance, with the availability of real-time location and current absorption of the trams, a quasi-real-time monitoring of the tramway network could be achieved. Moreover, the proposed GIS platform could serve as a hub to integrate additional data sources and applications.

Acknowledgements

I wish to thank all the technicians at Infra.To with whom I could collaborate during the development of this thesis, for their availability and interest towards this work, which has been a valuable learning experience.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The purpose of this thesis is twofold: firstly, it has been developed as part of an ongoing collaboration between Politecnico di Torino and Infra.To, the company managing the infrastructure of the tramway network in Turin, with the goal of delivering a software tool that can carry out simulations on the traction electrification system, to assist the maintenance as well as the design phase of future works on the network itself. The possibility of computing currents both in normal operating conditions and in the presence of faults supports the dimensioning of the network as well as the calibration of protective devices against faults. Specifically, the problem of distinguishing fault currents from the currents due to the normal operation is essential to the safety of the system: to this end, the knowledge of the minimum possible values of fault currents is of great importance.

On the other side, in the framework of the research of the Energy Center Lab at Politecnico di Torino, the wish is to integrate this tool in a large project that aims to build a comprehensive mapping of the energy consumption in the city, in which the tramway network plays a significant role. In terms of project choices, this has particularly driven the project choices towards the criteria of openness and flexibility, through the usage of open-source software and a web-based architecture.

1.2 Turin tramway network

An overview of the characteristics of the Turin tramway network can be found in [1] and [2]. Throughout the city there are around twenty conversion substations, where AC medium voltage is rectified and transformed to 600 V DC. From these substations, a system of underground cables, consisting of positive cables, feeder cables and interconnections made of copper bars brings the electrical energy to the

overhead contact system (OCS), from which tram vehicles absorb power through their pantographs. Current then flows away from the vehicles through the wheels and the tracks; a system of negative cables connected to the tracks brings it back to the substations, closing the circuit. To limit the stray currents dispersed by the tracks into the ground, negative cables are not connected to the substation grounding system. A simple diagram representing the network structure is shown in Figure 1.1. Substations are modelled with their Thévenin equivalent circuit; all cable elements, as well as OCS conductors and faults, are modelled as resistances; trams are instead modelled with their Norton equivalent circuit. Lastly, tracks are modelled as distributed parameters lines, with a longitudinal resistance and a shunt conductance to ground, to model the dispersion of stray currents. [2] is taken as reference for the values of the physical parameters.

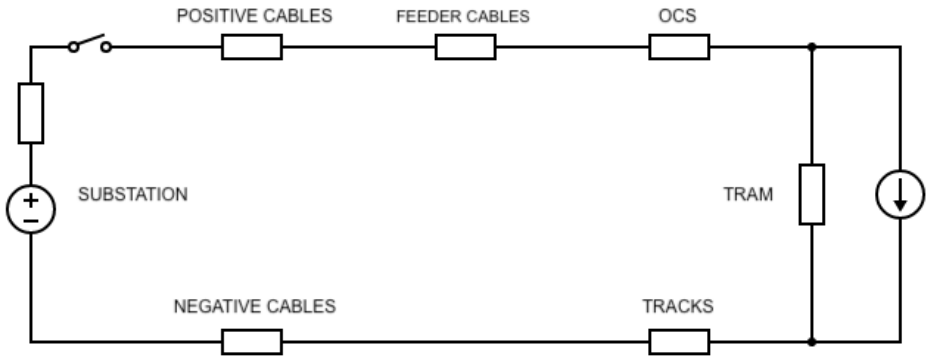

Figure 1.1: Diagram of the tramway network electrification system.

The network is divided into separate zones: each one is powered by a single primary substation and one or more backup ones. In turn, each substation can feed up to 6 or 7 OCS zones. Each zone is protected by an extra-rapid dc circuit breaker, whose overcurrent settings depend on the size of the zone, as well as on the forecast number of vehicles circulating in it. Across the OCS, this division is ensured by the presence of zones dividers, made of insulating materials.

In general, a tramway network differs from railway ones in that it is a meshed network, with many interconnections even over small distances. Moreover, even if the OCS zones are independent from each other, the tracks and negative cables compose a unique meshed network across the whole city. For this reason, tramway networks are significantly complex from the topological point of view: in turn, this affects the complexity of the computations required to simulate them.

The network can be configured in different ways, either controlling the switches of the individual zones (located at the substations), or by physically modifying the copper junctions present across the network, which determine how positive cables

and feeder cables are connected to each other.

1.3 Thesis outline

The rest of this thesis is structured as follows: first, Chapter 2 presents the state of the art of railway systems simulators and of the usage of Geographic Information System (GIS) in the management of power distribution networks; subsequently, it motivates the project choices and it details the requirements of the proposed software tool. Then, Chapter 3 provides some background on the main technologies employed in the development of the application. Chapter 4 explains the architecture of the application: how each component is structured and how they interact with each other; as a deepening, Chapter 5 focuses on the functionalities of the simulator. Chapter 6 discusses the results obtained with the simulator: it describes the features of the MapStore interface, a validation performed with LTspice and a comparison with results obtained in [1] on the same network. Finally, Chapter 7 concludes the thesis outlining some possible expansions of the work carried out.

Chapter 2

State of the art

The literature review starts with the study of existing software tools dedicated to the simulation of railway systems. None of these were found to address explicitly the scenario of tramway systems. Nonetheless, it is worth taking them into consideration, in comparison to the requirements of this work. Specifically, Section 2.1 reports an overview of the structure of the existing commercial simulators, while Section 2.2 provides a brief description of each one of the examined tools and a comparison among them in terms of key features. Then, Section 2.3 contains a discussion on the usage of GIS software for the management of electrical power networks. Section 2.4 draws conclusions based on the previous findings, pointing out the limits of existing solutions with respect to the use case of this thesis. Finally Section 2.5 defines the requirements of the project.

2.1 Railway power network simulators

A presentation of the principal components of a railway power network simulator is given in [3]. The authors describe how a railway system is composed of three main parts: 1) the signalling system; 2) the power system; and 3) the traction equipment.

1. The signalling system ensures the safety of train movement and the regulation of traffic flow. It has two main functions: to ensure a safe distancing between vehicles and to ensure a safe operation of trains at junctions and track intersections. Depending on the requirement of different systems, the implementation can be very different.

2. The power network can either be DC or AC. The DC is preferred in metro systems where lines are shorter, speeds are lower and less power is required; DC power is derived from the utility AC supply through transformer/rectifier

substations. In these cases, simulation can be limited to the DC side. When AC is used to carry power over long distances, booster transformers or auto-transformers are used to improve the transmission efficiency.

3. Traction drives must provide a flexible speed control. DC motors meet this requirement easily. AC induction motors have become more popular since the introduction of advanced high-rating thyristors, which can handle variable-voltage and variable-frequency input.

Still according to the authors of [3], the main processes to be simulated are: 1) the train movement; 2) the traction power supply; 3) the traction drives; and 4) the power network solution.

1. Usually, trains are modelled with an object oriented approach. The network structure is represented as a graph, i.e. a collection of nodes and a collection of links joining the nodes. Each of them is itself an object, and their various properties are encapsulated in the data structure of the object. The simulation may be either time-based or event-based. In the first case, train movement is evaluated at predetermined time intervals. This can give a very detailed account of the system evolution, but can be computationally demanding. On the other hand, event-based models simulate the system as a sequence of events, such as arrival and departure at stations. Trains interact via the signalling system and this allows an event to trigger other ones. If on one side this can reduce the computational complexity, avoiding the calculation of the exact movements, on the other it introduces the issue that train movements are not updated synchronously.

2. The electrified railway line can be schematized as a large circuit where substations are voltage sources and trains are moving loads. The voltage seen by the train can vary with time, and this can have an effect on its performance. Therefore the voltage at nodes must be evaluated at regular intervals. In the case of a DC feeding system, where the overhead contact system supplies the current and the rails are used as return paths, all the conductive components can be assumed linear. Efficient algorithms are needed in order to solve the matrix equations.

3. A model for the traction equipment is required to provide the current demands and traction effort output according to the power network calculation and the input parameters for train movement. The model can either be voltage-sensitive or not, depending on the required accuracy. The traction systems are represented with a V-R model, where V and R can vary with the speed and the operational mode of the train. The model is used to generate look-up tables of V and R for the different conditions.

4. This is the most time-consuming step of the simulation. The power network simulator can either be integrated with the train movement simulator or be considered as an independent entity. In the first case, there is a feedback of information from the power network solution to the movement simulation: if the power network is not able to provide the required power for a train to sustain the desired speed, this has an impact on the successive step in the train movement simulation. In the second case, the train movement is simulated on its own, based on constraints as time-tables and speed limits; then, the train positions and corresponding velocities (from which power or current requirements can be derived) are used as input to the power network simulator at each instant. This feedback of information, which is present in almost all commercial packages, is needed if the performance limits of the whole system are to be thoroughly investigated.

   The problem can be formulated in a matrix form using nodal analysis. As described in [4], the main issues to be addressed are: 1) the large dimension of the admittance matrix; 2) the dynamic topology; 3) the regenerative braking.

2.2 Comparison of existing software for the simulation of electrical train power supply systems

There are numerous commercial software tools for the simulation of railway power supply systems. Some of them are part of broader platforms for traffic planning, which implement a variety of functionalities that go beyond the analysis of the power system; some others, instead, focus solely on the electrical infrastructure.

In the first case, the train movement simulator is embedded in the traffic planning tool and exchanges information with the power network simulator. It is the case, for instance, of the traffic planner OpenTrack and the power network solver OpenPowerNet.

In general, the train movement simulator takes the following input data:

- the infrastructure, i.e. the network topology, which includes the signalling systems;
- the timetable data, i.e. the schedule that trains should respect;
- the rolling stock data, i.e. the models of the different types of vehicles to be simulated.

At each time step of the simulation, the differential equations of the train motion are solved to calculate the tractive power required to respect the timetable. This,

together with the train positions, which determine the electrical network topology, becomes the input of the power network simulator, which solves the circuit model verifying the feasibility of the power requirement. As already mentioned, in most software packages, the result of the analysis is fed back to the traffic simulator and conditions the actual motion of the vehicles.

As reported in Table 2.1, this feature is implemented by the majority of the examined tools.

Software	OpenPowerNet [5]	eTraX [6]	Sidytrac [7]	TracFeed [8]
Manufacturer	OpenTrack Railway Technology Ltd	ETAP	Siemens	Adtranz / Balfour Beatty Rail AB
DC networks	✓	✓	✓	✓
AC networks	✓	✓	✓	✓
Geospatial representation		✓		
Graphical track editor	✓	✓	✓	
Retroaction to train driving dynamic	✓	✓	✓	✓
Short circuit calculations	✓	✓	✓	✓
EM field calculation	✓	✓	✓	
Optimization of feeding configuration	✓	✓	✓	
Animation during simulation	✓	✓		

Software	SIGNON Suite [9]	FABEL [10]	TTS/SIMON PowerLog [11]	TPSS [12]
Manufacturer	SIGNON	ENOTRAC AG	ÅF-Industriteknik	Academic
DC networks	✓	✓		✓
AC networks	✓	✓	✓	✓
Geospatial representation				
Graphical track editor	✓	✓		✓
Retro-action to train driving dynamic	✓	✓		✓
Short circuit calculations	✓	✓	✓	✓
EM field calculation	✓	✓		
Optimization of feeding configuration	✓	?		
Animation during simulation			✓	

Table 2.1: Feature comparison of existing software.

In the following, three of the analysed tools are discussed in more detail.

2.2.1 OpenTrack and OpenPowerNet

Begun in the mid-1990s as a research project at ETH, Zurich, OpenTrack is a railway simulation tool largely used in the industry. It allows the simulation of a variety of train systems, such as high speed rail, freight, commuter, tram and underground systems.

It allows to perform several tasks such as headway calculation, timetable construction, rolling stock studies.

OpenPowerNet is its corresponding electric network solver: it is supposed to determine if a traffic solution suggested by OpenTrack is allowed by the power system. In OpenTrack, the complexity of the propulsion model can be configured, so that it fits different scenarios; for instance, the efficiency factors of the propulsion equipment can be considered constant or voltage dependent.

The feedback of information to the planning software is of particular importance, especially when DC network are involved, since they have high load dynamics, for the following reasons: 1) power losses increase with decreasing voltage; 2) when voltage drops, current and power limitations are activated, with an impact on the driving dynamics; 3) the network voltage influences the capability of the system to recover braking energy.

2.2.2 eTraX

eTraX is the software by ETAP for the simulation of rail power systems. It focuses only on the traction power systems. Among the examined solutions, eTrax is the only one implementing a geospatial view aside from the one-line diagram, to represent the network. The two views are automatically kept in sync and both can be used to design the network, to input data and to view the real time animation of the vehicles during the simulation.

2.2.3 TPSS

Outside the world of commercial software, a simulator developed in a PhD project is TPSS (Train Power System Simulator), discussed in [12]. The models, differently from the previous examples, are fully disclosed. The ultimate goal of the project is to suggest an investment planning: in this framework, it is useful to perform a high number of simulations and a number of simplifications are acceptable. Moreover, in the same project an approximator, called TPSA (Train Power System Approximator) was developed as a neural network. Given the infrastructure and a certain level of train traffic, it outputs the performance of the power network. This is done in order to further reduce the computational complexity of the simulations.

The train model consists in a curve which, for the available voltage and the desired speed, gives the required power.

The set of differential equations that give the voltages and power flows in the networks, as well as the acceleration (when it is positive) of the trains is solved as an optimization problem in GAMS (General Algebraic Modelling System), where the power unbalance in the network is minimized. When a train is braking or proceeding at constant velocity, its acceleration does not depend on the power system.

2.3 GIS and power distribution networks

The usage of GIS for the management of AC power distribution systems is well documented. Again, these entail significant differences from the DC tramway network under study, in terms of topology, equipment to model and analysis required. Still, a lot benefits are shared with the use case of this thesis and are worth being examined.

It should be noted that a lot of research has also been done integrating GIS in the context of city transport systems. However, in this area the focus is on the services offered rather than on its infrastructure, offering support for applications such as route planning, fleet management, traffic analysis. As a consequence, this is less relevant for the purpose of this thesis and it will not be investigated.

2.3.1 Benefits of GIS in power distribution networks

A good description of the role of GIS in the context of smart grids is given in [13]. Smart grids are essentially intelligent electric systems which allow to manage loads with more efficiency, provide a higher level of automation during service restoration and enable better interaction between energy providers and consumers, relying on the synergy between ICT technologies and electric transmission and distribution networks. To this end, a GIS integrates hardware, software and data to collect, manage, process and display geo-referenced information.

In general, through the visualisation of data on a map, GIS allows utility providers to better understand the physical and spatial relationships among network components and with the surroundings, in comparison to relying only on one-line diagrams or other abstract models of the network.

GIS is designed to integrate a variety of external data sources, from digital terrain models to land use data, weather data and so forth. Core business applications can then merge these sources with data served from SCADA systems, providing a unified picture for inspection and maintenance and for network analysis and planning.

In short, GIS can be the hub of all data sources and an efficient output to the users. In this way, it can improve a variety of tasks, such as:

- asset management, helping to determine optimal locations to install new equipment and build new facilities;
- network analysis, visualising either sensor data or simulation results on a map, thus overcoming the limitations of abstract representations of the network which lack geographic representation, as one-line diagrams;
- work-force automation, helping to schedule and dispatch utility crews, monitoring their location and status;
- outage management and situational awareness, thanks to the availability of real-time data, graphic outputs and web-based reporting, that improve monitoring;
- visualise historical events over time, thanks to the support of spatio-temporal databases [14];
- spatial load forecasting: according to [15], the integration with land use data and customer records improves the ability to predict the amount and the location of future load growth. This happens to be of particular relevance in developing countries, where the annual load growth is of the order of 10%.

2.3.2 GIS models for power distribution networks

As described in [14], since GIS is supposed to be the central hub that gathers all kinds of data sources, the problem of sharing data with other enterprise applications is crucial. Moreover, GIS should eliminate the need of maintaining multiple network models for planning.

Geographic data models are the abstractions used to represent the electrical distribution facilities: the ones provided by vendors (an example is GIS application POWERGIS®and the Distribution Network Analysis, Optimisation, Planning and Design Application POWERNET®, developed by Global Energy Consulting Engineers Pvt. Ltd. and described in [16]) are proprietary; utilities either have used them or developed custom ones. As a consequence, ad-hoc interfaces are often needed when data has to be shared. Indeed, the Open GIS Consortium (OGC) provides standards for the exchange of geographic data, having defined the Geographic Markup Language (GML), but lacks semantic models for the electric utility industry.

Still according to [14], the interface process can be improved by exploiting a GIS data model based on Common Information Model (CIM) developed by the International Electrotechnical Commission (IEC). CIM is defined in standards IEC 61970 and IEC 61968: based on UML, its goal is to describe electric networks, allowing easy exchange of information across different platforms and companies.

The authors of [17] propose also an extension to CIM to model railway electrified systems.

Thus, the geographic model provides a spatial representation of the components, allows the representation of network data on the map and supports a variety of GIS analysis functions, like spatial queries, thanks to the support of spatial databases, which allow to store geometry together with attributes. This is then mapped to a logical model which abstracts the network as a graph, where nodes can represent sources, switch nodes or other equipment, and edges represent feeder lines. [18] From this, a variety of algorithms can be applied to perform the analysis of interest. From the logical network, algorithms can derive automatically the one-line diagram of the network.

The connectivity rules can be defined either in the geographic model or in the logical one. For instance, [19] reports the development of an object-oriented model to represent power network in GIS based on the open-source desktop application framework uDig that allows to draw standard equipment and determine the connectivity among components. uDig also allows for customization with Eclipse Rich Client Platform (RCP) plugins. Instead, in [18] it is the logical network that implements the connectivity rules and describes the internal behaviour of the different components.

Finally, another application developed on an open-source GIS platform is [20]. In the thesis, the author makes use of the OpenGeo Suite, which is no longer maintained. However, some of its standalone components are: namely, PostgreSQL with the PostGIS extension as geospatial database, GeoServer to make geospatial data available through the web and a web client for visualisation based on ArcGIS, a popular commercial GIS platform developed by ESRI. Both PostGIS and GeoServer will be actually chosen for this work and will be described in Chapter 3.

2.4 Conclusions and project choices

Given this context, a number of reasons advise against the usage of commercial simulators in our use case. Firstly, their value lies in the fact that they allow for sophisticated dynamic simulations, with advanced physical models of traction systems: this is actually not needed for our purpose. The primary goal is rather to carry out static simulations, solving the network at a particular point in time. Transients need not to be taken into consideration and modelling traction systems as constant current sources with a parallel conductances will suffice. Secondly, railway networks are topologically simpler than the tramway network, in terms of aerial lines: without some testing, it is not clear whether these applications would support the different connectivity that the tramway network entails. Thirdly - and most importantly - being proprietary tools, they are not designed to be

easily customised nor integrated with other application components. Particularly, it can be assumed that they would not support the import of line drawings from AutoDesk AutoCAD, the editor currently used by Infra.To for the design of the network. Finally, they would not enable a fully web-based solution and they would be expensive.

Therefore, it seems sensible to develop an application which revolves around an open-source web-based GIS platform for the storage and the visualisation of geo-referenced data for both network, vehicles and other external sources. From this interface, a custom simulator will be accessed through a web API. It will process the network data, build an object-oriented electrical model of it and solve it applying nodal analysis, feeding the results to be displayed back in the GIS interface. As for the geometric model of the network, the simplest solution is to retain the one used in AutoCAD, preserving the layer definitions as much as possible. The topological model will be developed inside the simulator; it will be in charge of defining connectivity and it will be tailored to solve the specific analyses required in our use case.

2.5 Requirements

From this reasoning, the following requirements for the application are set:

2.5.1 GIS interface

Input

- interface with AutoDesk AutoCAD to import the model of the network; this is the editor which Infra.To already uses to draw the network elements (cables, substations, etc.) and its advanced features are needed when significant changes have to be made to the network;
- apply small changes to the network directly through the GUI (e.g. replace a cable, or edit its attributes);
- configure the substation switches to choose which zones are powered and from which point;
- position trams along the network, manually, through the GUI;
- position faults (i.e. short-circuits) along the network, manually, through the GUI;
- draw tramway lines to simulate multiple static scenarios in sequence.

Output

When simulating a static scenario:

- display a colour map of the network according to currents and voltages;
- display labels providing detailed information of currents and voltages;
- display summarised information for each substation, tram and fault point;

When simulating the time evolution of the network, in addition to the previous points:

- move along time to observe the output of each time step;
- show statistics of the whole simulation (i.e. max currents, min voltage).

2.5.2 Simulator

- simulate a single static scenario (i.e. solve the electrical network with some trams and/or some faults in a fixed position);
- simulate the normal evolution of the system as a sequence of static scenarios (i.e. simulate the tram movements along the lines);
- for a given zone, determine the point of maximum resistance, which allows to determine the lowest short-circuit currents.

Chapter 3

Enabling technologies

3.1 RESTful web services

The Representational State Transfer (REST) [21] architectural style was developed as a model for the interactions within web applications. In this model, any piece of information is considered a resource and has a Uniform Resource Identifier (URI). The requirement is that resources are acted upon through a set of uniform interface semantics that allow to exchange resource representations (e.g. HTML or XML documents, JSON object, JPEG images, etc.). A common implementation of this architecture is the HTTP protocol with its GET, POST, PUT and DELETE methods. REST interfaces must be stateless, i.e. each request must include all information to be fully understood, independently of previous requests [22].

3.2 Internet Geographic Information Systems

Internet GIS can be described as network-based systems that use Internet to access and distribute information, tools and services [23]. These applications generally implement the client-server model, where the first makes requests and the second fulfils them. This model consists of three primary elements: presentation (the user interface), logic (the processing) and data (a database). Through a browser, the user can enter the URL corresponding to some spatial data, interrogating a web server, which will query the database for the requested data and return it to the browser, after having reformatted it in a way it can understand. Shifting away from the concept of desktop GIS, that used to stimulate the development of proprietary programs and data models, Internet GIS encouraged the adoption of open standards to ease the sharing of data and the interoperability of services among different providers.

3.2.1 OGC standards

In this context, it is essential that information exchanged is machine independent: to achieve this, the Open GIS Consortium (OGC), which is the primary standards issuing entity for the GIS community, has defined GML, an open-source markup language based on XML. GML objects encode together feature geometries (e.g. Line, Points, Collections thereof), properties (numeric, text, boolean fields), spatial reference (coordinate reference system). Among the OGC Web Service (OWS) standards, a series of protocols that can be implemented as WebAPIs are defined [24]. Here two of the most used ones are described: the Web Map Service and the Web Feature Service.

Web Map Service (WMS) WMS provides an HTTP-based interface for requesting map images from a geospatial database [24]. The request defines the layers and the area of interest. The response is one or more map images (in formats as JPEG, PNG, etc.) that can be displayed in a browser application. Parameters as the transparency of the image can be specified. In particular, the following operations are supported:

- GetCapabilities: it lists the data that is available on the server in an XML document;
- DescribeLayer: it provides a description of a given dataset;
- GetMap: it returns the map image described in the request.

Web Feature Service (WFS) WFS provides interfaces for querying and modifying geospatial information [25], [26]. In particular it offers data manipulation operations on geographic features, as: creating, deleting, updating and getting features, with the possibility of specifying spatial or non-spatial filters. Among the supported operations, the principal ones are:

- GetCapabilities: receiving this request, the WFS server returns an XML document that describes the feature types it can offer and the operations defined on each one of them;
- Describe Feature Type: it provides a schema description of the supported feature types;
- GetFeature: it returns the GML data of the features specified in the request;
- LockFeature: it sets a lock that prevents a feature from being edited;
- Transaction: it allows to create new features, as well as to update and delete existing ones.

3.2.2 Apache Tomcat

Apache Tomcat is a free, open-source HTTP web server that allows to host and run Java web applications [27]. Tomcat is developed and maintained by an open community of developers which the supported by the Apache Software Foundation. It is released under the Apache License 2.0 license.

3.2.3 GeoServer

GeoServer is an open-source software server that allows to share and edit geospatial data [28]. Started in 2001 by The Open Planning Project (TOPP), a non-profit technology incubator based in New York who envisioned the development of an easily accessible Geospatial Web analogous to the World Wide Web, it is specifically designed for interoperability and the use of open standards.

It offers a Web administration interface for extensive configuration, from which it is possible to handle the publication of vector and raster layers from a variety of data sources, including databases like PostGIS, MySQL, Oracle and many more.

3.2.4 PostGIS database

PostGIS is a spatial database extension for the object-relational database PostgreSQL, which enables it to store data for GIS [29]. It is an open-source project started by Refractions Research and continues to be developed by a group of contributors led by a Project Steering Committee.

3.2.5 MapStore

MapStore is an open-source WebGIS framework developed by GeoSolutions that allows to create, manage and share maps, integrating remote content from providers such as Google Maps, OpenStreetMap, Bing or other servers compliant to OGC standards (e.g. GeoServer) [30].

As a standalone product, MapStore is a powerful and interactive geospatial WebGIS, with access to geospatial data warehouses through the most common standards and advanced spatial analysis capabilities; it can also be expanded adding plugins. As a framework, it can be integrated in a project to develop custom applications.

The main technologies used in the front-end are ReactJS and Redux. As for the map libraries, MapStore is map agnostic, meaning it supports both OpenLayers (which is the default map library for desktop devices), LeafletJS (used by default for mobile devices) and Cesium 3D viewer.

React React is a JavaScript library for building user interfaces, i.e. the view of the application [31]. This is done through the composition of components, which are written with JSX, a sort of composition of HTML and Javascript code. Components can be passed properties and can have a state. The basic idea is that whenever the state of a component changes, it gets re-rendered, without the need of updating the whole web page.

Redux Redux is a state container for Javascript applications [31]. It allows to centralize the application's state and logic, helping to write consistent and easy to test applications. The glue library React Redux makes it possible to bind Redux state to React components. It defines a single, global Store that contains the whole state of the application. Actions can be dispatched to the store in order to change the state: when this happens, reducer functions are called. Reducers are passed the action and the current state: from these, they produce a new state. React components can be connected to the Redux store, notified of state changes so that they can be re-rendered. Two types of middlewares, called Redux thunks and Redux Observables, allow to perform asynchronous data calls, like Ajax requests.

3.3 Python

Python is a popular general-purpose programming language that offers efficient high-level data structures and a simple approach to object-oriented programming [32]. It is an interpreted language, meaning program development does not require compilation and linking. It offers a wide range of libraries to approach countless tasks, from scientific and numeric applications, to machine learning, web development and so forth. In the following, a brief description of some of the libraries used in the project is given.

GeoPandas GeoPandas is an open-source project for the processing of geospatial data in Python. It extends the datatypes used by pandas to enable spatial operations on geometric types. It leverages the shapely library to perform geometric operations. It also depends on fiona for file access and matplotlib for plotting. In this way, it allows to perform spatial analysis in Python without having to rely on a spatial database.

NumPy NumPy is a library for scientific computing in Python. It supports multidimensional array objects, which provide better performance compared to working with native Python objects as lists.

CherryPy CherryPy is a minimalist Python web framework. It allows to develop web applications with an object-oriented approach.

3.4 Docker

Docker is an open platform for developing, shipping and running applications that allows to separate the application from the infrastructure by providing a loosely isolated environment called container [33]. Containers embed everything is needed to run the application, with no dependencies on the host environment, but are lightweight: compared to hypervisor-based virtual machines, they are faster to be deployed and require less space. In this way Docker platform increases both the portability of the application and its scalability, making it easier to scale up or tear down services as needed.

Docker implements a client-server architecture, where the server is the Docker daemon, which is responsible for managing images, containers, networks and volumes, while the client is the tool through which users can interact with Docker, sending REST API requests to the daemon.

Docker has a public registry called Docker hub where users can pull existing images. Private registries can also be configured to store and share images. Images are read-only templates of instructions for creating a container. They are built through Dockerfiles, which are text files that contain a list of commands for Docker to build the image. Each instruction in the Dockerfile corresponds to a layer of the image. Altogether, they completely define the configuration of the container that will be created. New images can also be built from the running state of a container. When a container is removed, changes to its state disappear: to persist storage beyond the life-cycle of a container, either volumes or bind mounts can be used. In bind mounts a path in the container file system is mapped to a path on the host file system (this solution is therefore dependent on the host file system and it is discouraged in production). Using volumes, a path on the container file system is assigned a name (i.e. the name of the volume) and stored on the host in a way that is internally managed by Docker, and can be referenced through the name. Moreover, volumes allow to share data among containers.

Docker Compose is a particular client that allows to work with applications consisting of a set of containers, coordinating their deployment through a configuration file (.yml), called compose file, which defines the list of services and the configuration which would usually be provided at run time. For instance, the docker-compose file is used to define the following elements: the port mapping, i.e. how some ports in the container are mapped to the host; the docker volumes, which allow to persist data outside the container filesystem; environment variables, that allow to change some parameters with which the images are built; finally, it

allows the creation of a Docker network through which containers can communicate. Beside this general configuration file, each service has its own Dockerfile which specifies how to build each single image.

Chapter 4

Application architecture

4.1 Overview

The system is organised as a multi-container Docker application, orchestrated with Docker Compose. A graphical representation of its services is reported in Figure 4.1.

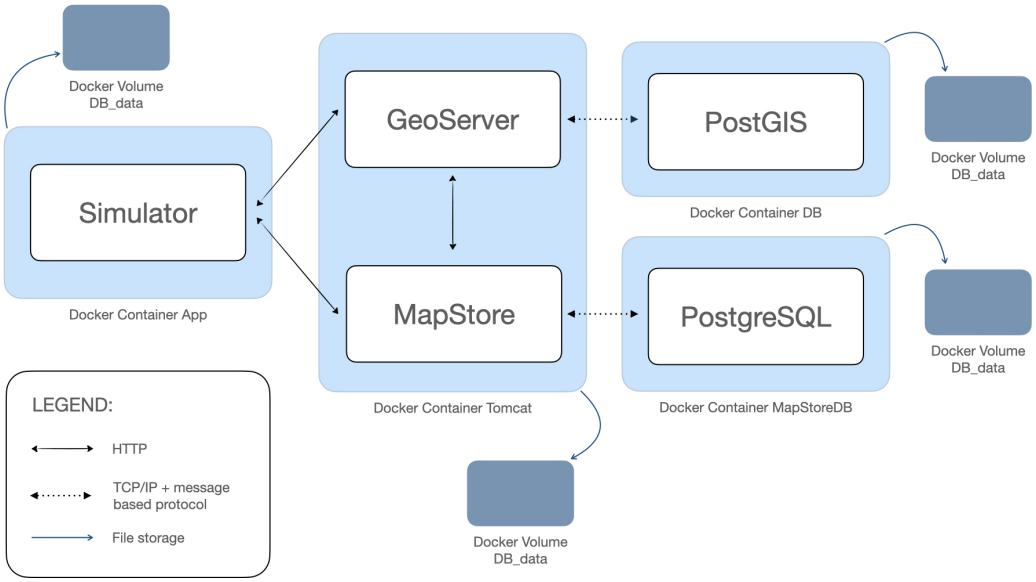

Figure 4.1: Architecture of the application.

With Docker Compose, the highest level of configuration is provided in the `docker-compose.yml` file, which defines the list of services (i.e. of containers) included in the application. In this case, the services are:

- `app`, which is the CherryPy server that hosts the simulator;
- `tomcat`, an Apache Tomcat server that hosts both the GeoServer and MapStore applications, which are distributed as web application resource (WAR) files;
- `db`, a PostGIS database that stores all the layers published in GeoServer;
- `mapstore_db`, a PostgreSQL database that stores the configuration of MapStore, including both users and maps.

From the point of view of communications, the containers are all inside a Docker network that allows them to reference each another using service names as host names. All communications among the containers, with the exception of those involving the front-end of the MapStore application, take place in this way. Whenever this is the case, access is provided directly to the internal container ports (and not to the host ones to which container ports can be mapped). For instance, the simulator fetches layers from GeoServer at: `http://tomcat:8080/geoserver/turin_tramway_network/wfs`, where 8080 is the container port exposed by Tomcat. Similarly, GeoServer connects to the PostGIS database using host `db` and port `5432`, while the MapStore backend connects to its own database with host `mapstore_db` and port `5432`.

Additionally, each container exposes some ports that are mapped to host ports, which are accessible from outside the Docker network. Specifically, the `app` service maps the container port `8084` to the host port `8084`; the `tomcat` service maps its container port `8080` to the host port `8085`. Therefore, `8085` is the port at which both GeoServer and MapStore can be accessed from the browser. Furthermore, whenever the MapStore front-end code references the other containers, to connect to GeoServer and to the simulator, it does so by specifying their ip addresses and the host ports.

As for data storage, each container mounts a Docker volume that allows to persist critical data beyond the life cycle of the containers themselves.

In addition to the docker-compose configuration file, each service has its own Dockerfile which specifies how the image should be built.

4.2 App service

This image is created starting from the `python:3.9.5` image. On top of that, a list of libraries is installed. Namely, they are: cherrypy, cherrypy-cors, shapely, requests, pandas, numpy, geopandas. Then, the source code of the simulator is loaded, together with the default configuration file and the (empty) default switches configuration file. An environment file is provided to load some environment variables: specifically, the authentication key to access the Cherrypy server

functionalities and the key that the CherryPy server uses to contact GeoServer in the tomcat container. Moreover, a Docker volume, called `app_data`, is mounted, to persist the configuration file and the Circuit object file, which is saved and updated when simulations are carried out. Finally, the main script, `App.py`, that loads the Cherrypy server, is called: the application is deployed and the server starts listening to requests.

4.3 Db service

This image is built from the image `postgres:12`. On top of it, the PostGIS extension (version 3) is installed. An initialisation script is then called, which does the following: firstly, it creates a database called `turin_tramway_network`; secondly, it creates a user called `geoserver`, which is then used by GeoServer to connect to the database, use it as a data store and publish its content; thirdly, in the database it creates three schemas: `input`, `network` and `results`, whose content will be described in detail in the rest of this section.

An environment file loaded in the initialisation script defines the credentials to the database, as well as the coordinate reference system to be used for all the tables, which in this way can be easily modified. The `/var/lib/postgresql/data` directory is mounted on a volume called `db_data`, so that all data is persisted on the host filesystem, outside the Docker container.

4.3.1 Input schema

The `input` schema contains three tables: `trams`, `faults` and `lines`. They host the input to be used for the simulations.

Trams The structure of the `trams` table is reported in Table 4.1.

Field	**gid**	**model**	**serial_number**	**line**
Type	serial	varchar	varchar	varchar

Field	**current**	**included**	**notes**	**geom**
Type	real	boolean	varchar	geometry (Point, crs)

Table 4.1: Fields of table `input:trams`.

- **gid**: it is the primary key of the table, i.e. a unique identifier for each record;
- **model**: it indicates the model of the tram;

- **serial_number**: it indicates the serial number of the vehicle;
- **line**: it indicates the tramway line on which the vehicle is located;
- **current**: it is the amount of current absorbed by the tram, expressed in ampere;
- **included**: it indicates whether the tram is to be included in the simulation or not;
- **notes**: it contains comments for the record;
- **geom**: it contains the spatial coordinates of the vehicle, relative to the specified coordinate reference system.

Faults The structure of the `faults` table is reported in Table 4.2. The description of fields already mentioned in previous tables is omitted when the meaning is the same.

- **resistance**: it is the value of the resistance that fault has, expressed in ohm.
- **geom**: it contains the spatial coordinates of the fault, relative to the specified coordinate reference system.

Field	**gid**	**resistance**	**included**	**notes**	**geom**
Type	serial	real	boolean	varchar	geometry (Point, crs)

Table 4.2: Fields of table `input:faults`.

Lines The `lines` table structure is given in Table 4.3:

Field	**gid**	**number**	**direction**	**notes**	**geom**
Type	serial	varchar	varchar	varchar	geometry (MultiLineString, crs)

Table 4.3: Fields of table `input:lines`.

- **number**: it is the identifier of the tramway line;
- **direction**: it identifies the travel direction along the line;
- **geom**: it contains the coordinates of all the segments that make up the line. It is a collection of LineString elements.

4.3.2 Network schema

The `network` schema includes ten tables that together compose the different layers of the tramway network infrastructure. In general they are in correspondence with the layers defined in the AutoCAD drawing file, with some exceptions that will be described in detail. Since the AutoCAD layers are labelled in Italian, the database tables keep the Italian names. The translation in English is always reported in the following discussion.

Positive cables, feeder cables, OCS cables Positive cables, feeder cables and overhead contact system (OCS) cables each have their own table. Here there is a first difference compared to the AutoCAD layers, where each OCS zone was stored in a separate layer. In the database they are stored together, but the `Layer` field preserves the original layer information (e.g. `LA_20`, `LA_55`, where LA stands for "Linee Aeree" - Italian for OCS.). This simplifies the layer management in the maps, where an excessive amount of items to import would be cumbersome. When needed, filtering can be applied to identify all the OCS features related to a single zone.

A common structure is shared among the tables `cavi_positivi` (positive cables), `cavi_alimentazione` (feeder cables) and `linee_aeree` (OCS cables) and it is reported in Table 4.4.

Field	**gid**	**Layer**	**zone**	**EntityHandle**
Type	serial	varchar	varchar	varchar

Field	**Linetype**	**length**	**notes**	**geom**
Type	varchar	real	varchar	geometry (LineString, crs)

Table 4.4: Fields of tables `network:cavi_positivi`, `network:cavi_alimentazione` and `network:linee_aeree`.

- **Layer**: it is the name of the AutoCAD layer to which the object belongs;
- **zone**: it indicates the network zone to which the cable belongs;
- **EntityHandle**: it is a unique identifier assigned by AutoCAD: in the simulator it is used, together with the `Layer` field value, as a unique id for each object;
- **Linetype**: it contains information on the size of the section of the cable: for some layers it is expressed numerically in squared millimetres, for others as a code that has a mapping in the configuration file;

- **geom**: it contains the coordinates of the line that represents the cable. Each cable record is structured as a single LineString, with no discontinuities.

Negative cables, tracks Being cables themselves, these two layers have the same fields of the other cable types discussed above, with only one exception: in these two tables, called `cavi_negativi` (negative cables) and `binari` (tracks), the `zone` field is not defined, since the 'negative' side of the network forms a unique mesh that does not reflect the division into zones of the 'positive' side of the network. The structure of the tables is given in Table 4.5.

Field	**gid**	**Layer**	**EntityHandle**
Type	serial	varchar	varchar

Field	**Linetype**	**length**	**notes**	**geom**
Type	varchar	real	varchar	geometry (LineString, crs)

Table 4.5: Fields of tables `network:cavi_negativi` and `network:binari`.

Junction boxes This table contains the junction boxes elements. Originally, the AutoCAD layer with this name (`cassette` in Italian) included also the substations from which positive cables start. In the database these are stored separately in the `cabine` table (Italian for "substations"), since from the point of view of the simulator they are handled in a totally different way. Indeed, junction boxes have no functional role in the simulator: they are just included to be displayed on the map, together with the text labels that are attached to them (which are stored in a separate table themselves). Table 4.6 reports the structure of this database table.

Field	**gid**	**Layer**	**label**
Type	serial	varchar	varchar

Field	**EntityHandle**	**notes**	**geom**
Type	varchar	varchar	geometry (LineString, crs)

Table 4.6: Fields of table `network:cassette`.

- **label**: some text that describes the junction box, which is useful to be displayed on the map;
- **geom**: it contains the square or rectangular box that identifies the perimeter of the junction box.

Substations As mentioned above, this layer is not present in the AutoCAD drawing file, but it is created from the `cassette` (junction boxes) layer, with the subset of junction boxes that represent the substations from which positive cables start. The structure of the table, which takes the Italian name `cabine` (equivalent to "substations") is reported in Table 4.7.

Field	**gid**	**Layer**	**name**	**code**
Type	serial	varchar	varchar	varchar

Field	**EntityHandle**	**notes**	**geom**
Type	varchar	varchar	geometry (LineString, crs)

Table 4.7: Fields of table `network:cabine`.

- **name**: it is the name of the substation, written as a label in AutoCAD;
- **code**: an additional string label that identifies the substation, along with the name;
- **geom**: it contains the square box that identifies the perimeter of the substation.

Junctions This table hosts the junctions of the circuit, i.e. the copper bars, physically hosted in a junction box in the ground, that connect cables together, allowing for different configurations. Table 4.8 gives the structure of the table, which retains the Italian name `lame` (equivalent of `junctions`).

Field	**gid**	**Layer**	**zone**	**EntityHandle**
Type	serial	varchar	varchar	varchar

Field	**length**	**notes**	**geom**
Type	real	varchar	geometry (LineString, crs)

Table 4.8: Fields of table `network:lame`.

Labels Finally, there are two tables that contain respectively the labels of the OCS zones and of the junction boxes.

The first table, called `labels_la` has one point-like feature for each zone in the circuit, reporting the number of the zone: in AutoCAD it corresponds to the layer `LA_ETI` (where `ETI` is short for "etichette" - Italian for "labels").

The second, called `labels_cassette` has one or more points for each junction box, each containing a text field that indicates the code of the junction box, the street name, or some other identifier. The reason why there is a dedicated layer and the labels are not all included in the layer `cassette` itself is that when the AutoCAD drawing file (DWG) is exported to the DXF format, the junction boxes objects are exploded and the GeoPandas library is not able to retrieve the association between the geometric object of the perimeter of the junction box (stored in the layer `cassette`) and the point-like features of the labels, which get stored in the generic layer `0`. Since the labels are mapped as points outside the perimeter, trying to associate them with the junction boxes (i.e. with the perimeter objects) could generate errors when more boxes are close to each other. The simplest way to reproduce the labels is then to keep them in a separate layer. Furthermore, this ensures that they can be easily activated or not on the map. Table 4.9 reports the common structure of the two labels tables.

Field	**gid**	**Layer**	**text**	**geom**
Type	serial	varchar	varchar	geometry (Point, crs)

Table 4.9: Fields tables `network:labels_la` and `network:labels_cassette`.

- **text**: it is the content of the label, the text that should be displayed on the map;
- **geom**: it contains the coordinates of the point at which the label should appear.

4.3.3 Results schema

Lastly, the `results` schema defines tables to store the results of the simulations. In this case, the division into tables does not reflect the `network` schema, but another approach is used.

Links To further simplify the display of the results on the map, all cable objects are grouped in one unique table, called `links`, that contains only the links where there is a flow of current. This is done to limit the amount of result features stored for each simulation. The table structure is given in Table 4.4.

- **Layer**: it is always the name of the AutoCAD layer to which the object belongs; in this table it is of particular importance to distinguish e.g. positive cables from OCS cables, which are stored together;

Field	**gid**	**Layer**	**EntityHandle**	**zone**
Type	serial	varchar	varchar	varchar

Field	**length**	**current**	**voltage_drop**	**power**
Type	real	real	real	real

Field	**notes**	**Date**	**geom**
Type	varchar	timestamp with time zone	geometry (LineString, crs)

Table 4.10: Fields of table `results:links`.

- **current**: it contains the amount of current flowing through the link, expressed in ampere;
- **voltage_drop**: it contains the voltage drop that takes place on the link, expressed in volt;
- **power**: it contains the amount of power flowing through the link, expressed in watt;
- **Date**: it contains the timestamp of the simulation. This field is fundamental to distinguish records of different simulations;
- **geom**: it contains the coordinates of the line that represent the cable. Each cable record is structured as a single LineString, with no discontinuities.

Nodes This table contains the list of the nodes of the circuit that are either start or end points of a link in which there is a flow of current. The structure of the table is given in Table 4.11.

Field	**gid**	**zone**	**voltage**
Type	serial	varchar	real

Field	**notes**	**Date**	**geom**
Type	varchar	timestamp with time zone	geometry (Point, crs)

Table 4.11: Fields of table `results:nodes`.

- **voltage**: it contains the voltage level of the node, expressed in volt;
- **geom**: it contains the coordinates of the node, in correspondence to an end-point of a link.

Trams The `trams` table contains the results for all the trams included in the simulation. Table 4.12 reports its structure.

Field	**gid**	**model**	**serial_number**	**line**
Type	serial	varchar	varchar	varchar

Field	**zone**	**current**	**voltage**
Type	varchar	real	real

Field	**notes**	**Date**	**geom**
Type	varchar	timestamp with time zone	geometry (Point, crs)

Table 4.12: Fields of table `results:trams`.

Faults Similarly, the `faults` table contains the results for all the faults included in the simulation. The structure is reported in Table 4.13.

Field	**gid**	**resistance**	**zone**	**current**
Type	serial	real	varchar	real

Field	**voltage**	**notes Date**	**geom**	
Type	real	varchar	timestamp with time zone	geometry (Point, crs)

Table 4.13: Fields of table `results:faults`.

- **resistance**: as in the corresponding table in the `input` schema, it indicates the resistance of the fault.

Max resistance nodes Lastly, this table is dedicated to store the results of the simulation task where, for a given zone, the point of maximum resistance is evaluated. Table 4.14 reports its structure.

- **zone**: the zone for which the point of maximum resistance is evaluated;
- **substation**: the name of the substation from which the zone is powered;
- **fault_resistance**: the value of resistance (expressed in ohm) given to the test fault;
- **equivalent_resistance**: the value of equivalent resistance present between the generator and the fault point, expressed in ohm;

Field	gid	zone	substation	fault_ resistance
Type	serial	varchar	varchar	real

Field	equivalent_ resistance	min_ current	voltage_ drop	power
Type	real	real	real	real

Field	notes	Date	geom
Type	varchar	timestamp with time zone	geometry (LineString, crs)

Table 4.14: Fields of table `results:max_resistance_nodes`.

- **min_current**: the amount of current that flows at the fault point, expressed in ampere. It is the smallest value across the zone, for the given powering substation;
- **voltage_drop**: the value of voltage drop across the fault resistance, expressed in volt;
- **power**: the amount of power that flows through the fault, expressed in watt;
- **geom**: the coordinates of the fault point at which the minimum fault current is found;

4.4 Tomcat service (GeoServer)

The `tomcat` service is built on top of the official tomcat image. In addition, the Dockerfile simply contains the command to copy into the container the `geoserver.war` file, together with the pre-configured data folder and the expanded `geoserver` folder, that contains some extensions not included in the basic installation. Namely, these plugins are: the key authentication module, the CSS Styling extension and the WMTS multidimensional extension. The version used for the application is GeoServer 2.19.0. By default, when the container starts, the Tomcat server is started and the GeoServer application is deployed.

As for the configuration of GeoServer, a first aspect to consider is authentication. By default, GeoServer comes with an administrator user, called `admin`. An additional user, called `simulator-user` is defined: this is to allow the simulator app to authenticate to the server; the user is associated to a newly defined `EDITOR` role, for which read and write privileges on all layers are granted.

To authenticate themself, each user has their own credentials: username and password. This is the so called basic authentication system. Besides this, an

alternative that is compatible with the simulator app, which is a simple OGC client that cannot handle any kind of security protocol, is needed. This is addressed by the key authentication extension, which allows to generate a key for each user: when the key is appended to an URL request, GeoServer recognises the user and fulfils the request, if it is within the privileges of the role associated to that user. Clearly, this authentication technique is exposed to security token sniffing and it should only be used with HTTPS connections: this has not been implemented yet and should be object of future work.

On the contrary, technique does not work with the MapStore client, which, too, requires access to the server. To allow MapStore to make authenticated requests, GeoServer should be configured so that it recognises MapStore users, according to the procedure described in [34]. However, this functionality has not been implemented yet.

For the purpose of the application, this GeoServer instance features a single workspace, called `turin_tramway_network`. For each workspace, a list of stores can be defined: they act as data sources from which GeoServer can load raster or vector layers to publish. A store can be a directory of shapefiles, a CSV file, a schema of a geospatial database, or an external Web Feature Service. In this case, three stores are defined, which correspond to the `input`, `network` and `results` schemas of the `turin_tramway_network` PostGIS database described in the previous section. For all three schemas, GeoServer connects to the database as user `geoserver` (who has read and write privileges): the service name `db`, specified in the `docker-compose.yml` file, is used to identify the host of the database and connect to it through its internal container port `5432`.

Once a store is connected, GeoServer can publish one layer for each table included in the schema of the store: in this way it can publish the content of the three input tables, the ten network tables and the five results tables detailed in Section 4.3. Figure 4.2 shows a screenshot of the interface with the list of published layers.

In addition to the layers, a list of styles is defined and associated as default to the different layers. Styles are defined in CSS. For the input layers, they specify special icons for trams and faults, so that they can be easily recognised on the map; for the network layers, they provide different colours for each layer and configure the display of substation names and OCS zones; for the results layers, they colour the links differently based on the current field value, displaying small arrows that indicate the direction of the flow.

This whole configuration is loaded when the `tomcat` image is built. Afterwards, the GeoServer web user interface allows to customise all aspects, from security to the publication of the layers and their styles. Namely, Geoserver data directory is mounted on a Docker volume, that ensures its persistence beyond the life-cycle of the container. For reference, the complete documentation for GeoServer 2.19.0 can be found at [35].

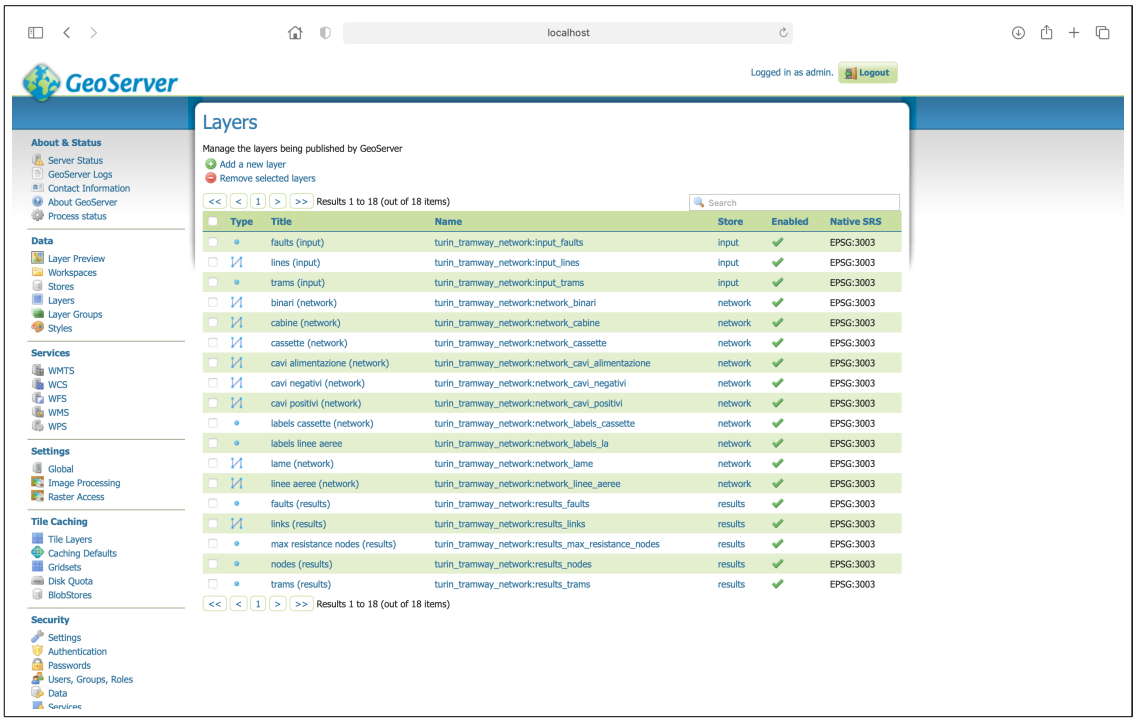

Figure 4.2: Layers list from the GeoServer web interface.

4.5 Tomcat service (MapStore)

In the same `tomcat` container described in Section 4.4, the MapStore application is deployed as well. The previously described Dockerfile includes also a line that imports the `mapstore.war` file and, separately, the `mapstore` folder, from which some configuration files can be edited.

The standard installation of MapStore includes two users, one called `admin`, another called `guest`. Once they are logged in, users can setup their "context", which is a map environment with a specific configuration of plugins. Once inside the map screen, through the "catalog" feature the user can search for layers, in WMS, WFS and other formats, and add them to the map. Once they are added, layers appear in the panel on the left side of the screen. A screenshot of this interface can be seen in Figure 4.3. A complete description of the features offered by the interface can be found in the MapStore User Guide, available at [36].

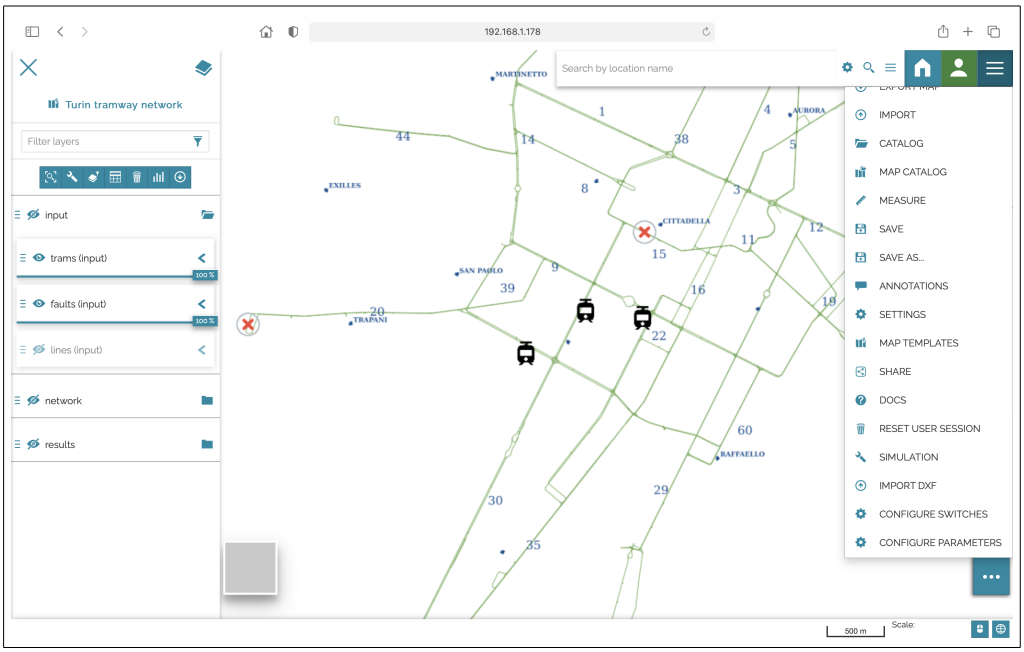

Figure 4.3: Map view of the MapStore interface: the left-side panel shows the list of layers that have been added to the map.

4.5.1 Additional plugins

The front-end of the MapStore application is organised as a composition of plugins. This makes it modular and rather easy to be expanded with new functionalities. The version of MapStore included in the application contains four plugins in addition to

the ones included in the distributed version. As a lot of the built-in plugins, these can be enabled or disabled from the MapStore GUI itself, during the configuration of the context. They are called `DxfImporter`, `Simulation`, `SwitchesConfig` and `SimulationConfig`.

The architecture of the four plugins is essentially the same. All four of them add a button in the burger menu, located in the top-right corner of the map page. When selected, each one opens a window through which the user can interact with the Simulator application. Each plugin defines a window component, connected to the Redux store, which is activated when the menu button is pressed. Embedded in the window component there is a form, written with the Formik library, which provides the actual interface with the simulator.

DxfImporter plugin Of the four, the first plugin to be used is the `DxfImporter` which allows to upload a DXF file to the CherryPy server. Its interface is displayed in Figure 4.4. When the application is loaded and the plugin component is mounted, it downloads the configuration file of the simulator with an AJAX GET request. The form has a box to select a DXF file to upload, browsing through the filesystem, and a field where the Coordinate Reference System parameter of the configuration file can be edited. When a file is selected and the form submitted, the plugin makes two AJAX requests of type POST to the server, the first with the updated configuration file, the second uploading the DXF file itself. When the upload is complete, the server automatically starts processing it, creating the circuit model, uploading the layers to GeoServer and creating the switches configuration file. A simple animation in the window indicates when the import process is complete and when it is safe to launch a simulation.

SwitchesConfig plugin The `SwitchesConfig` plugin allows to modify the status of the switches that enable the voltage sources in the network. It shows a list of all the substations present in the network and, for each substation, a list of the switches corresponding to all the zones powered by the given substation. The interface can be seen in Figure 4.5.

When the application is loaded and the plugin component mounted, the switches configuration file is downloaded with an AJAX GET request from the CherryPy server. If it is empty, when the plugin window is opened, a message is displayed, suggesting to upload a DXF file first. If it is not empty, it shows the switches panel, with check marks reporting their current status, which can be edited. A 'save' button at the end of the list allows to submit the form, triggering an AJAX POST request that uploads the updated switches configuration file to the server.

It should be underlined that the switches panel is dynamically displayed based on the content of the switches configuration file, which is generated upon the import of a DXF. Successive imports would overwrite the configuration file and

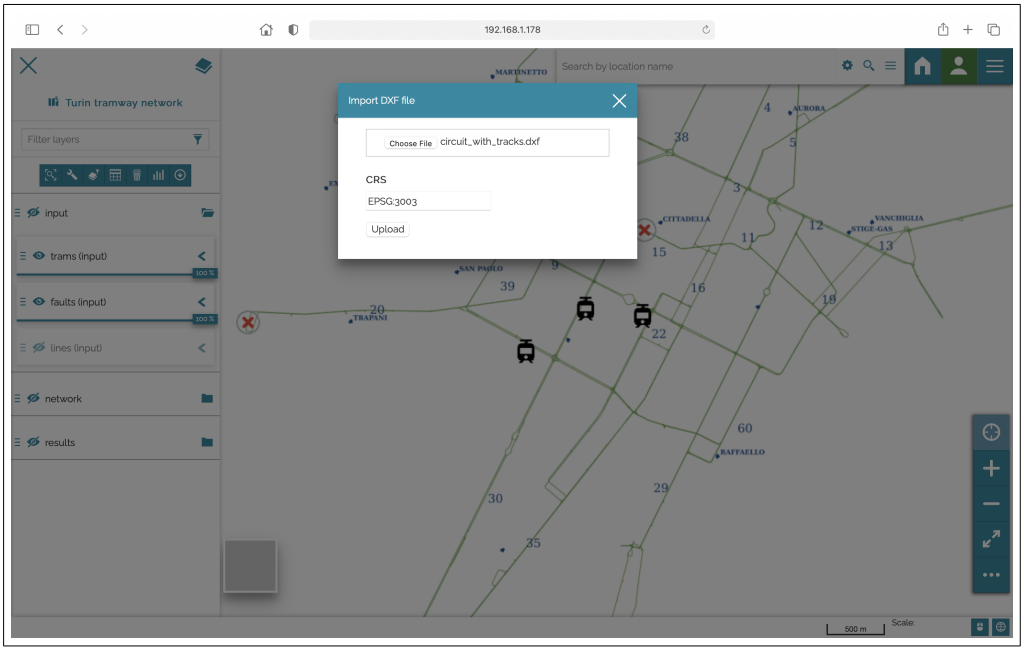

Figure 4.4: Interface of the `DxfImporter` plugin.

the switches panel would be updated accordingly. By default, all switches are set to false, i.e. the corresponding voltage sources are turned off.

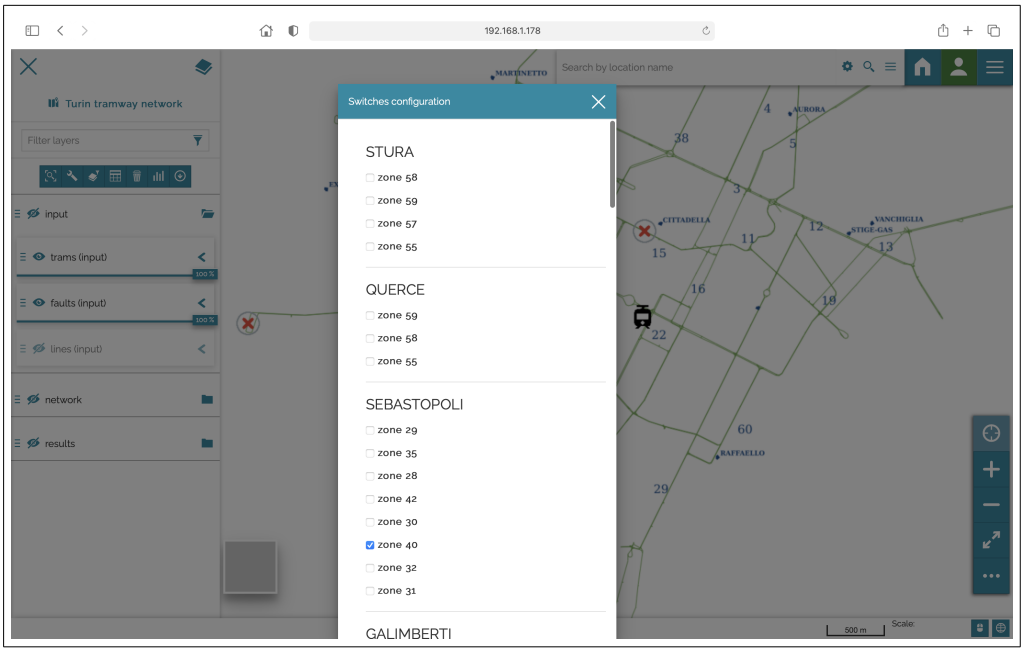

Figure 4.5: Interface of the `SwitchesConfig` plugin.

SimulationConfig plugin The `SimulationConfig` plugin allows to interact with all the editable sections of the configuration file, which are: 'Generators', 'Substations', 'Cables', 'Trams', 'Faults' and 'GeoServer'. For the most part, these contain parameters that affect how the circuital model of the network is built; the last section instead allows to configure the connection of the simulator to GeoServer. The list of editable fields is shown in Figure 4.6, Figure 4.7 and Figure 4.8. Again, when the application is loaded and the plugin is mounted the configuration file is downloaded; when the plugin is opened the current values of the parameters are displayed as default values in the form. The 'save' button, i.e. the form submission, triggers a POST request to the server that uploads the updated version of the configuration file.

Simulation plugin Lastly, the `Simulation` plugin is the one that allows the user to launch simulations. The plugin interface is shown in Figure 4.9. When the window is opened, the configuration file is downloaded with an AJAX request. The form allows to choose the task of the simulation, which is inserted in the URL of the GET request to the simulator, while all the other parameters are stored in the

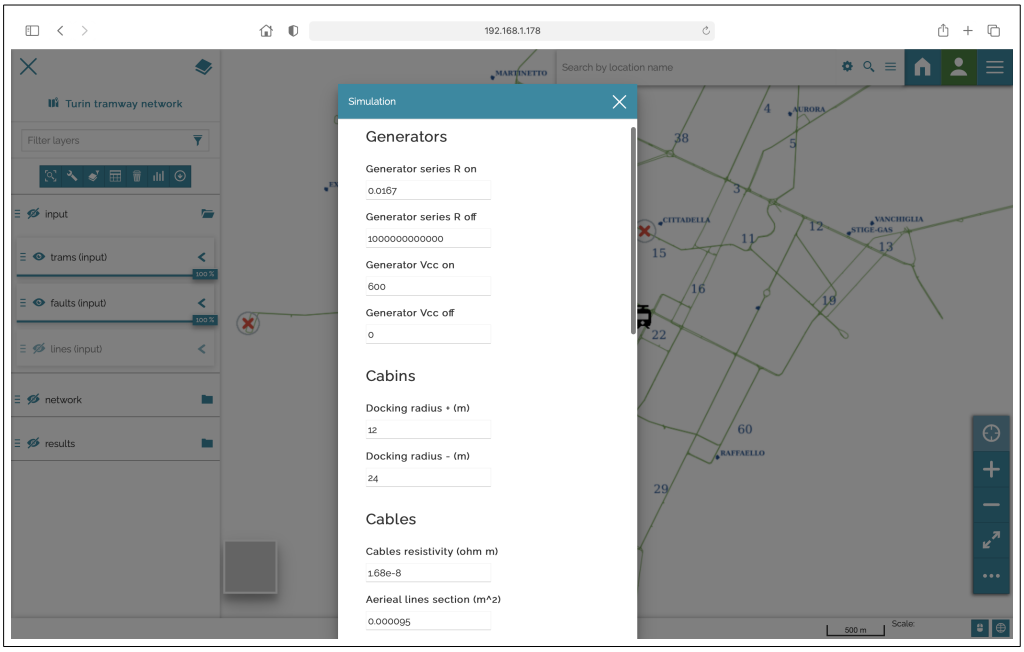

Figure 4.6: Interface of the `SimulationConfig` plugin (1/3).

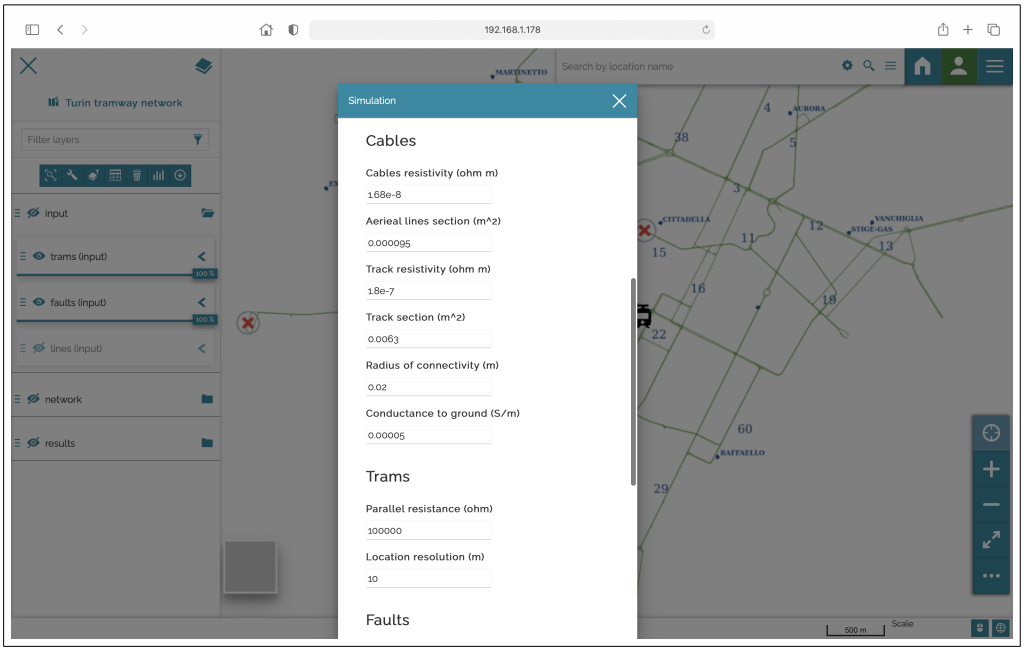

Figure 4.7: Interface of the `SimulationConfig` plugin (2/3).

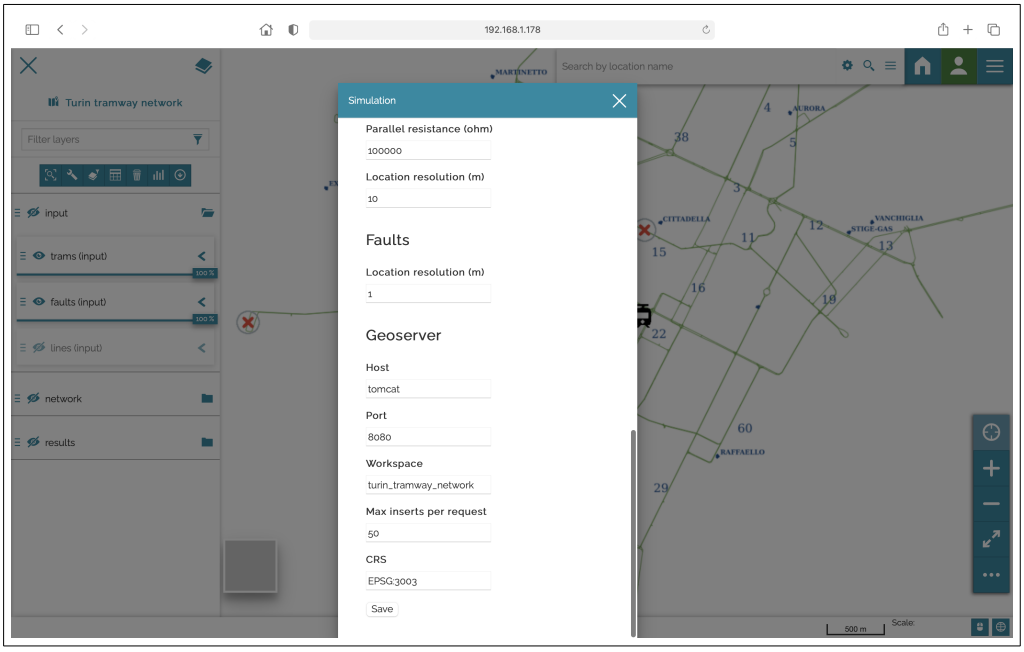

Figure 4.8: Interface of the `SimulationConfig` plugin (3/3).

configuration file. When the launch button is pressed, first the configuration file is uploaded with a POST request, then a GET request calls the simulator, starting the specified task, which can either be a simulation or the computation of the point of maximum resistance for a given zone.

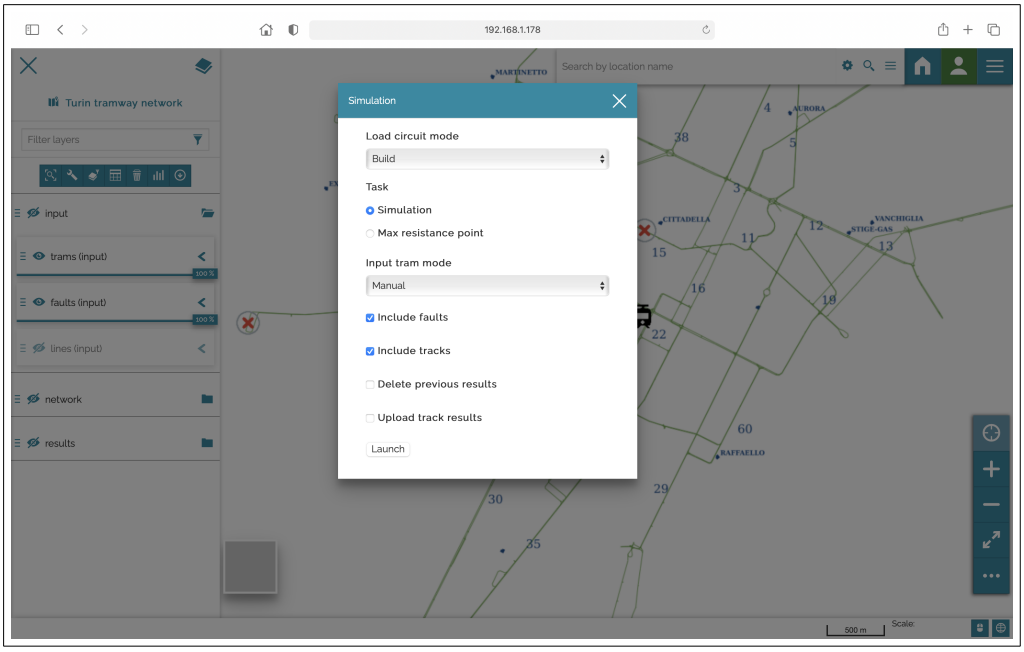

Figure 4.9: Interface of the `Simulation` plugin.

4.5.2 Connectivity with other application components

The plugins are part of the MapStore front-end code, that runs in the browser. As already described, these plugins need to perform various AJAX calls to the CherryPy server hosting the simulator; therefore, they need to be aware of its URL. Ideally, this should be indicated in the `localConfig.json` file (located in the `mapstore/config/` folder), which stores the configuration of all the MapStore plugins, and which can be edited after MapStore has been built and deployed. However, despite several trials, it was not possible to successfully make these requests when passing the simulator URL from the configuration files. At the moment, the simulator URL is hard-coded in the source code of the plugins. The fact that in this case it is not possible to reference it with its service name, as provided by the Docker network, is an additional inconvenience. As of now, the MapStore application has to be built embedding the URL for the simulator: further work should be carried out to solve this issue. On the other hand, the connectivity with the database, hosted in the `mapstore_db` service does not present the same

problem. The database configuration is specified in another configuration file, `geostore-datasource-ovr.properties` (located in the `mapstore/WEB-INF/classes/` folder), which can be edited after the deployment, and, being run on the server, it is able to exploit the Docker network host naming system. It specifies the connection as: `geostoreDataSource.url=jdbc:postgresql://mapstore_db:5432/geostore`.

4.6 Mapstore db service

The `mapstore_db` service is built upon the `postgres:13.2` image. In addition, two scripts provided by MapStore are run; they create a database called `geostore` and the tables required to store users, maps and other configuration of MapStore. The procedure is explained in detail in [37]. Lastly, this enables the integration of MapStore and GeoServer, which allows MapStore users to authenticate to GeoServer as well. However, as already mentioned, this is not part of the current configuration, since it caused issues with the management of the key authentication service required for the GeoServer `simulator-user`.

Chapter 5

Network simulator

This chapter describes the structure of the simulator used to build and solve the circuital model of the Turin tramway network. As already anticipated, the simulator is a Python application exposed on the web through a CherryPy web-server, hosted in a Docker container.

5.1 CherryPy WebServer

The server hosts a single application, called `App`, which is written as a Python class with the same name. The `App` class defines five objects as attributes:

- the configuration file, a JSON object that stores all the settings for the application;
- the switches configuration file, a JSON object that stores the list of the switches in the network, along with their status;
- an object of the `DxfImporter` class, that implements all the methods to import a DXF file;
- an object of the `Simulator` class, that implements all the methods of the simulator;
- a dictionary containing a key-value pair that should be used in the URL as authentication token to access the server functionalities.

Following the REST approach, the server exposes the following methods:

- `GET`: it allows to call the simulator, to retrieve the configuration file or to retrieve the switches configuration file;
- `POST`: it allows to upload the configuration file, the switches configuration file (both in JSON format), or a DXF file with the network data.

5.2 DXF importer

The `DxfImporter` class allows to: process a DXF file; build an electrical model of the tramway network as an object of the `Circuit` class; and upload to GeoServer the network layers, so that they become available to be visualised on a map, as well as to be accessed by the simulator.

When a DXF file is uploaded through a POST request, a sequence of operations is performed: first, the configuration file is loaded and the DXF file is read; then some pre-processing tasks are performed and the circuit model is built. Finally, the newly created network layers are uploaded to GeoServer.

5.2.1 Reading the DXF file

A DXF file is not meant to be understood when opened directly with a text editor. In Python, it can be read with the library GeoPandas, that imports it as a GeoDataFrame. Each record of the GeoDataFrame has the following attributes:

- Layer: indicates the AutoCAD layer to which the record belongs. A complete list of the layers present in the file is reported in Table 5.1;
- PaperSpace: it is empty and it is discarded;
- SubClasses: it contains information on the AutoCAD object class it belongs to; it is not needed and it is discarded;
- Linetype: when the geometry is of type LineString, it specifies a property of the line: in the layers corresponding to cables, it gives the section of the cable in mm^2; in the other cases it is ignored;
- EntityHandle: it is a unique identifier given by AutoCAD to each record, which will be used to identify each feature;
- Text: it can contain a text label attached to the feature;
- Geometry: it contains the geometry of the feature, which can be of type Point or LineString.

5.2.2 Preprocessing the DXF file

Firstly, the GeoDataFrame is assigned a coordinate reference system, specified in the configuration file; secondly, the fields `PaperSpace` and `SubClasses` are dropped, since they do not contain relevant information; thirdly, the records are divided into distinct GeoDataFrames according to their layer; the complete list of layers is reported in Table 5.1, which shows also which ones are discarded (a lot of layers

are not relevant to the purpose of the simulator); finally, the layers corresponding to the different zones of the overhead contact system (OCS) (e.g. `LA_02`, where `LA` stands for Linee Aeree, Italian for OCS and `02` is the zone number) are merged into a single layer, named `LINEE_AEREE`. This is done to reduce the overall number of layers to be handled.

Layer	**Geometry**	**Retained**
CASSETTE	GeometryCollection	✓
LAME	LineString	✓
CAVI_POSITIVI	LineString	✓
CAVI_NEGATIVI	LineString	✓
CAVI_ALIMENTAZIONE	LineString	✓
LA_*	LineString	✓
BINARI	LineString	✓
LA_ETI	Point	✓
0	Point	✓
PALI_ATM_ALIM	GeometryCollection	
USCITE_ALIM	GeometryCollection	
GIUNTI	LineString	
_cavi-appoggio	LineString	
_Varie_Ruby	LineString	
CODCAVI	Point	
ASSE_CAVIDOTTO	LineString	
PALI_ATM	GeometryCollection	
PLANIMETRIE	Point	
QUOTE	LineString	
DEFPOINTS	Point	
SEZIONI	LineString	
CAVIDOTTI	LineString	
TESTO_CART	Point	
RETE_5000	Point	
BOTOLE	LineString	
TESTI	Point	
CAVI-POS_NEW	LineString	
MOTRICI	GeometryCollection	
FS_Definitivo	LineString	

Table 5.1: Layers in the DXF file.

5.2.3 Building the circuit layers

Creating equipment objects At this point, the different GeoDataFrames are processed and each record is transformed into a Python object. Records of the GeoDataFrames positive cables, feeder cables, OCS cables, negative cables and tracks are stored as objects of the `Cable` class; records of the junctions GeoDataFrame are stored as objects of the `Junction` class; finally, records of the junction boxes GeoDataFrame are stored as objects of the `JunctionBox` class. As already mentioned in Chapter 4, this layer (originally named `cassette`) actually contains, other than the proper junction boxes, also the blocks which correspond to the substations. These two kinds of objects are distinguished based on their size: the substations are squares of side 15 m, while junction boxes never exceed a side length of 3 m. Based on this, a list of objects of the `Substation` class is created. All these objects are stored as lists into the `Circuit` object.

Determining connections Once all the features have been scanned, connections must be established. As they are extracted from AutoCAD, the features do not contain information on which other elements they are connected to. In fact, all the classes mentioned up to now (`Cable`, `Junction`, `JunctionBox`, `Substation`) constitute simply the physical model of the network. The connectivity model, instead, is implemented with two different classes: `Node` and `Link`.

First, the `Junction` objects are processed: for each of them, an object of the `Node` class is created.

Once this is done, the cables are scanned (only those of layers positive cables, feeder cables and OCS cables: at this stage tracks and negative cables are ignored). For each of them, the following algorithm is executed: first, the start point of the cable is considered. If it is in the neighbourhood (with a tolerance of 2 cm) of an already defined node, that node is assigned as start node. Otherwise, a new node is created with the coordinates of the start point. The same is done for the end point of the cable. Once the start and end nodes are defined, an object of the `Link` class is created. Each link contains references to both its start and end nodes; similarly, each node object contains references to all the links which either enter or exit it.

In addition, `Link` objects embed the `Cable` object from which they have been created: through this association, each one defines its own resistance as:

$$R = \rho \cdot \frac{l}{A} \tag{5.1}$$

where ρ is the resistivity of the cable, l its length and A its section.

Assigning zones At this point, the information on the zones of the circuit is only contained in the `Layer` field of the cables belonging to the OCS (e.g. `LA_19`).

By exploring the graph made of links and nodes, a zone is assigned to each link; the zone attribute should be unique, since each zone should be electrically isolated from the others. The only exception was found for zones 10 and 49, which appear to be connected in parallel. This information has then be confirmed by Infra.To. It should be pointed out again that zones are not defined for negative cables and tracks, which have not been considered yet.

The definition of the zone is of particular importance for the positive cables. These are the ones that exit from the substations, the first portion of the path that leads to the OCS. Since the substation features contain no data as to which zone they power, this key piece of information is determined in the following way: the end points of all positive cables are scanned. When one of these points is found to be in the neighbourhood of a substation (considering a radius of 12 m from the centre of the substation), it means that this substation actually powers the zone of that positive cable. Therefore, an object of the `Generator` class is created, embedded in the `Substation` object.

Creating the switches configuration file In the end, by scanning the list of substations and, for each of them, the list of its generators, the switches configuration file is created and saved as a JSON file. This file is exposed by the server and the status of the switches can be toggled from the GUI in MapStore.

5.2.4 Uploading layers to GeoServer

The last operation to perform is the upload of all the layers to GeoServer. To accomplish this, the `DxfImporter` uses the `GeoServerClient` class, that allows to connect to GeoServer using its REST API and the WFS protocol. It can be used to retrieve layers from the server (as done by the `Simulator` class), even applying filters, to upload layers (as in this case), to update specific values, or to delete records.

When a new DXF file is imported, the old content of the layers is deleted first, ensuring that only the latest version of the network is present.

5.3 Simulator

The simulator operates under the assumption that a network file has already been imported in the DXF format and that the corresponding layers have been published to GeoServer.

The simulator allows to perform two different tasks: either a simulation, or the determination of the point of maximum resistance for a given zone. Considering the simulation task, two main possibilities are given: the first is to perform a single

static simulation, the second to perform a series of static simulations, where tram vehicles move along a predetermined path.

5.3.1 Single static simulation

When the simulator is launched specifying the "simulation" task and the "manual" tram mode, firstly it loads the configuration file, which includes all parameters for the simulation. Secondly, it retrieves the input from GeoServer. In this scenario, the input is given by the layers `input:trams` and `input:faults`. Both represent point-like features that have been manually drawn on the map through the MapStore GUI, indicating the required attributes, like absorbed current for the trams and resistance values for the faults. They are downloaded from GeoServer and stored in GeoPandas GeoDataFrames. Then, the network layers (those that were uploaded as a result of the import of the DXF file) are downloaded from GeoServer and stored in GeoDataFrames, too. This is again accomplished with the support of the `GeoserverClient` class.

5.3.2 Building the circuit

At this point, some of the operations already described in Section 5.2.3 are executed again. Namely, all the GeoDataFrames are scanned and `Cable` and `Junction` objects are built; the difference is that this time the `network:cassette` (junction boxes) layer is not considered, but rather the `network:cabine` (substations) layer is: the distinction among them does not have to be repeated.

Then, `Node` and `Link` objects are built analysing `Junction` and `Cable` objects; zones are assigned and `Generator` objects are built. In this case, the generators are assigned a status, based on the switches configuration file, that determines whether they are active or not.

As already explained, each substation has a number of generators, one for each zone that it powers. Each generator embeds a link, that on the positive pole is connected to the positive cables of that zone that arrive at the substation. On the negative pole, all the generators of a given substation are connected together in a unique substation negative node. If tracks (and negative cables) are included in the model, all negative cables arriving at the substation will also be connected to that node. If tracks are not included in the model, the substation negative node will simply be the ground node, which would be therefore shared with all the other substations in the city.

5.3.3 Tracks and negative cables

Indeed, one of the parameters that has to be chosen when launching the simulation is whether to include the tracks (and, consequently, the negative cables) or not in the model. To include them means having a more accurate electrical model at the expense of a greater computational cost and thus time required to carry it out. Indeed, the `network:binari` (tracks) layer is by far the heaviest one, counting more than 24000 features. If included, they are processed in the following way: first, the tracks GeoDataFrame is scanned, `Cable` objects are created for each record and are stored in a separate list. Then, a new set of nodes and links is created: indeed, tracks cannot be connected with the 'positive' side of the network. The same algorithm previously described for the creation of standard nodes and links is used.

Then, the negative cables GeoDataFrame is scanned: as for the tracks, `Cable` objects are created and stored separately from the others. Then, nodes for the negative cables are created according to the following algorithm:

- if a cable is in the neighbourhood of a substation, it is connected to the substation negative node;
- otherwise, a connection is searched with other existing negative cables nodes. Here a larger radius is used, namely 2 m, because it happens that multiple negative cables arrive at a negative cables junction box, but these were not drawn in the AutoCAD file. In this case, they should be connected together;
- otherwise, a connection is searched with track nodes. Here a very large radius is used (100 m), since it happens that tracks are defined in long segments and the extremity of a negative cable is far away from its end points;
- otherwise, as a last option, a new negative cable node is created.

Once the connectivity model for tracks and negative cables is complete, an algorithm is implemented to reduce its number of features: by analysing the track links, it can be seen that almost two out of three are connected in series, i.e. there are over 16000 track nodes that have one link coming in and one coming out. These two links can be merged, by deleting the node in the middle, the two original links and adding a new one that has the geometry of the two original links merged together.

Finally, if tracks are included, the presence of stray current that is dispersed to the ground must be modelled as well. This is done by adding, for each track node, i.e. at both ends of each track link, an additional link to the ground, with a resistance that depends on the length of the track.

At this point, the circuit is built. The heaviest step is the one in which the tracks network is built. This alone takes around ten minutes, while all the other operations require less than a minute in total. Considering this, it is useful to store the `Circuit` object using `pickle`, a Python library that allows to store Python objects into binary files and load them when needed. In this way, unless some features in the circuit have been edited, the simulator can simply load this circuit file and reload the configuration file, thus saving the whole building time.

5.3.4 Inserting input

In the case of a single static simulation, the following operations are performed: first, the generators are updated, based on the status indicated in the switches configuration file. According to it, the system determines which are the active zones, and therefore which are the active links and nodes. Those that are not active (with the exception of tracks and negative cables links, for which the active property is not defined) are not included in the simulation. Again, this is useful to simplify the calculations and to speed up the solution of the network.

Then, faults are inserted, if there are any. Geometrically, they are defined as points. As attributes, they include the resistance value, a flag that allows to specify individually which ones should be included in the simulation and some other descriptive field. For each fault feature, an object of the `Fault` class is created; essentially, it embeds a link with the specified resistance value. If an already defined circuit node is found to be within a radius of 1 m around the position of the fault, the fault is connected to that node. Otherwise, the fault point is projected onto the nearest link, which is then divided in two, creating a new node in correspondence to the point of minimum distance to the fault. The fault is then inserted into this newly created node.

As for the negative pole of the fault, if tracks are included and the positive pole belongs to the OCS, the fault is connected to the nearest track node. If tracks are not included, or if the fault is associated to a node on positive cables or feeder cables, the negative pole of the fault will simply be the ground node.

The same procedure is then followed to add trams to the circuit: the only differences are that the positive pole of a tram can only be associated to a node that lies on the OCS, and that the tram link contains a current generator in parallel to the resistance.

5.3.5 Solving the circuit

Once all the input elements have been inserted, the circuit can be solved by means of nodal analysis. The first matrix to be built is the link admittance matrix $\mathbf{G}$. It has dimensions $L \times L$, where L is the number of links in the circuit, considering the

active links (i.e. those that belong to an active zone), plus, if tracks are included, all track links, negative cables links and the links that model stray currents from tracks to ground. Active links include both tram links and fault links, if they are connected to an active network zone. The matrix is diagonal and each entry $\mathbf{G}_{i,i}$ is the total conductance of the i^{th} link.

Vector $\mathbf{v}_s$ is defined as the voltage sources vector. It has dimensions $L \times 1$: it indicates the value of the voltage source contained in each link, in series to the link resistance. Generator links of active zones are the only non-zero entries of this vector.

Similarly, vector $\mathbf{j}_s$ is defined as the current sources vector. It has dimensions $L \times 1$ as well: it indicates the value of current sources contained in each link, in parallel to the link resistance. Tram links are the only non-zero entries of this vector.

Then, matrix $\mathbf{A}$ is defined as the reduced incidence matrix. It has dimensions $N \times L$, where N is the number of nodes in the circuit minus one. The reference node not included in the matrix is the ground node. In the model, N is the sum of all active nodes, plus - if tracks are included - all track nodes, negative cables nodes and substation negative nodes. Each entry $\mathbf{A}_{i,j}$ takes value: 1 if link j exits node i; -1 if link j enters node i; 0 otherwise. In this way, each column (i.e. each link) contains exactly one entry equal to 1 (on the row corresponding to the start node of the link), an entry equal to -1 (corresponding to the end node of the link) and all other entries are equal to zero.

Once all these matrices are defined, the nodal admittance matrix $\mathbf{Y}_n$ is computed as:

$$\mathbf{Y}_n = \mathbf{A}\mathbf{G}\mathbf{A}^T \tag{5.2}$$

The two unknown vectors $\mathbf{j}$, with dimensions $L \times 1$ and $\mathbf{e}$, with dimensions $N \times 1$ are defined, representing respectively the current at each link and the voltage at each node, i.e. the voltage difference relative to the reference (ground) node.

The KCL matrix formulation reads:

$$\mathbf{A}\mathbf{j} = \mathbf{0} \tag{5.3}$$

The vector of voltage drops across each link, $\mathbf{v}$, with dimensions $L \times 1$ is defined as:

$$\mathbf{v} = \mathbf{A}^T\mathbf{e} \tag{5.4}$$

The link currents vector can be expressed as the sum of the admittance matrix multiplied by the voltage drop across the links and the total current sources vector:

$$\mathbf{j} = \mathbf{G}(\mathbf{v} - \mathbf{v}_s) + \mathbf{j}_s \tag{5.5}$$

Rewriting (5.5) substituting $\mathbf{v}$ with the expression of (5.4), and multiplying left by $\mathbf{A}$, gives:

$$\mathbf{Aj} = \mathbf{AG}(\mathbf{A}^T\mathbf{e} - \mathbf{v}_s) + \mathbf{Aj}_s \tag{5.6}$$

where each side is equal to zero, by (5.3). Therefore, (5.6) can be rewritten in the form:

$$\mathbf{Y}_n\mathbf{e} = \mathbf{i}_s \tag{5.7}$$

where

$$\mathbf{i}_s \triangleq \mathbf{AGv}_s - \mathbf{Aj}_s \tag{5.8}$$

The unknown node voltage vector is then found as:

$$\mathbf{e} = \mathbf{Y}_n^{-1}\mathbf{i}_s \tag{5.9}$$

Finally, the link voltage vector $\mathbf{v}$ is found from (5.4) and the link current vector $\mathbf{j}$ is found from (5.5).

Of all these computations, the most expensive one is clearly (5.9), which requires the inversion of matrix $\mathbf{Y}_n$, which has dimensions $N \times N$.

To invert the matrix, first `numpy.linalg.inv` is tried. If tracks are included, this often fails, raising the error of non invertibility of the matrix. In this case, the exception is caught and the `numpy.linalg.lstsq` algorithm is used, a least square implementation that provides an approximate solution of the linear system, trying to minimise the total squared error.

The results (currents, voltages, power) are stored in the `Link`, `Node`, `Tram` and `Fault` objects. For each one of these classes, a GeoDataFrame is built, with only the features included in the calculations. Each record contains also a timestamp, that allows to distinguish features of different simulations. Lastly, the result GeoDataFrames are uploaded to GeoServer.

5.3.6 Lines simulation

An alternative to the single static simulation is to perform "lines" simulations, which can be specified by setting the "tram input mode" parameter to "lines". The difference lies in the tram vehicles input. Instead of the trams with their fixed positions drawn on the map, some lines (defined in the `input:lines layer`) are chosen. From each of these, a sequence of tram positions is generated, that correspond to a series of steps that are simulated in sequence. With the current implementation, this mode supports only a constant value of absorbed current for each vehicle. A future improvement would be to enable the import of a velocity/current profile that models the variation of current absorbed along the line.

The setup of this simulation requires to: download the lines features, create the set of tram coordinates for all steps, and then perform a single static simulation for every set of coordinates, each corresponding to a given moment in time. On one side, faults, if included, are kept statically throughout all the steps. Tram vehicles, on the other side, are inserted and removed after each step, using the set of locations defined for each step.

To remove a tram from the circuit essentially means removing the tram link: if an additional node was created for it, dividing an OCS link in two, the original link is restored, removing the two halves and the node in the middle, thus restoring the initial configuration.

When the results are uploaded to GeoServer, they look simply as a sequence of static simulations results, which differ in the position of the tram vehicles by a certain amount. Different parameters, such as the maximum number of steps, or the maximum spacing between steps, are specified in the configuration file and can be modified from the MapStore interface.

5.3.7 Max resistance point simulation

Alternative to the "simulation" task is the "max resistance point" task. This functionality allows to compute, for a given zone specified as input parameter, the point of maximum resistance. This corresponds to computing the lowest value of short-circuit current that can occur in that zone.

The build process of the circuit is the same described for the other simulations, with the only difference that here the switches are not setup based on the configuration file. Instead, they are initially all turned off, with the exception of one of the switches that control the zone to be simulated. Then, the OCS links are split into shorter segments, so that all links are shorter than a maximum length, specified as input parameter. Then, a fault is placed at one node of the zone and a standard simulation is carried out. After having stored the results and removed the fault from the circuit, the simulation is repeated moving the fault to another node of the zone. If the fault current is found to be lower than the previous one, the result is updated. After having iterated over all the nodes of the zone, the minimum value of fault current, corresponding to the point of maximum resistance, is determined. To have a more accurate result, at the expense of greater computational cost, it is sufficient to reduce the maximum allowed link length with which OCS links are split at the beginning of the algorithm.

The simulation is then repeated for all the different substations that can power the zone. Finally, for each substation a result record is uploaded to GeoServer, including the location of the point, the fault current value and the equivalent resistance seen from the substation.

Chapter 6

Results

This chapter presents some results obtained with the simulator. Section 6.1 shows the features offered by the MapStore interface; Section 6.2 presents a validation of the results obtained on zone 20 conducted with LTspice; finally, Section 6.3 compares some results related to zone 40 with the ones obtained by the simulator presented in [1].

6.1 Results visualisation in MapStore

Once a simulation is completed, the results are uploaded to GeoServer and can be accessed from the layers of the `results` group. Each simulation is timestamped: in this way, through the timeline feature of MapStore, it is possible to navigate through the results of different simulations, which are stored in the same layers.

As an example, we take a simulation on zone 20. This zone is powered by the two substations "Trapani" and "San Paolo". The scenario is the presence of a fault between the OCS and ground (in this case, tracks are not included), located at the farthest point from the two substations. The zone is powered in parallel by the two substations. Their voltage sources are set to $V_{cc} = 635\,\mathrm{V}$, with series resistance $R_{ser} = 16.7\,\mathrm{m\Omega}$. The resistance of the fault is set to $R_f = 1\,\mathrm{n\Omega}$. As already mentioned, tracks and negative cables are not included in the simulation. A screenshot of the MapStore interface is shown in Figure 6.1: the `links` and `faults` layers are visible.

By selecting the `links` layer and clicking on the map, a panel appears on the right side of the screen, displaying information on the selected features. Figure 6.2 shows an example of this for the `links` layer. The point in correspondence of the fault is selected, and the info of two cables appear: their original layer is the `LA_20` (i.e. they are OCS cables) and the `EntityHandle` field reads: `46323D_p0001` and `46323D_p0002`. Indeed, this was initially a unique cable, which was split in two

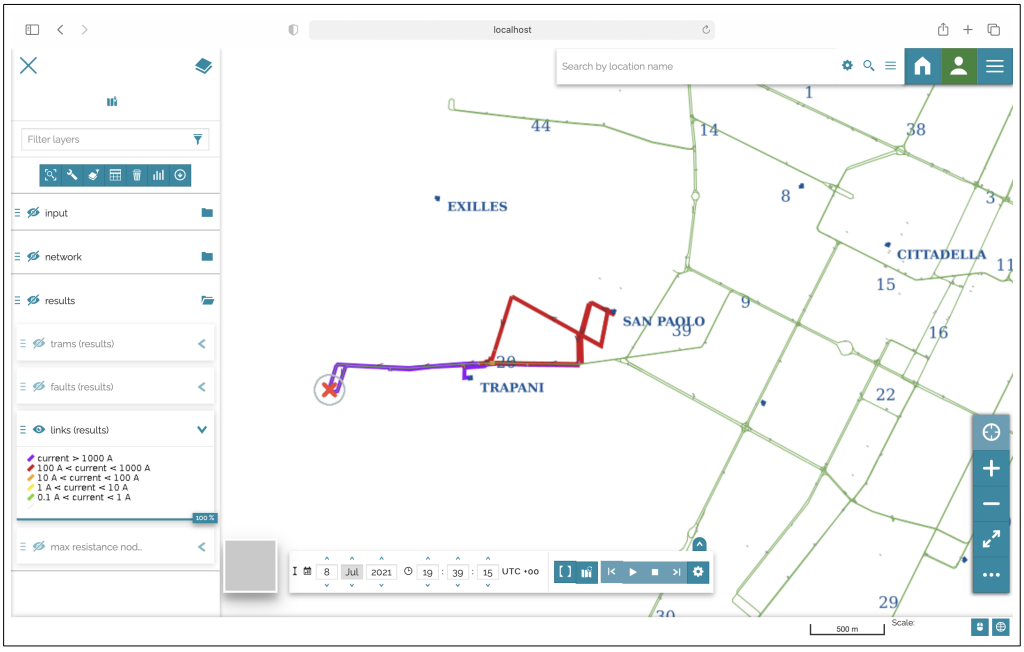

Figure 6.1: Visualisation of the results on MapStore.

during the simulation to insert the fault at that precise point; it was then stored as two separate features in the results layer. The arrows drawn along the link show how current flows from both directions into the fault and into the ground node. The two current values are respectively $I_1 = 3{,}623\,\text{A}$ and $I_2 = 3{,}643\,\text{A}$.

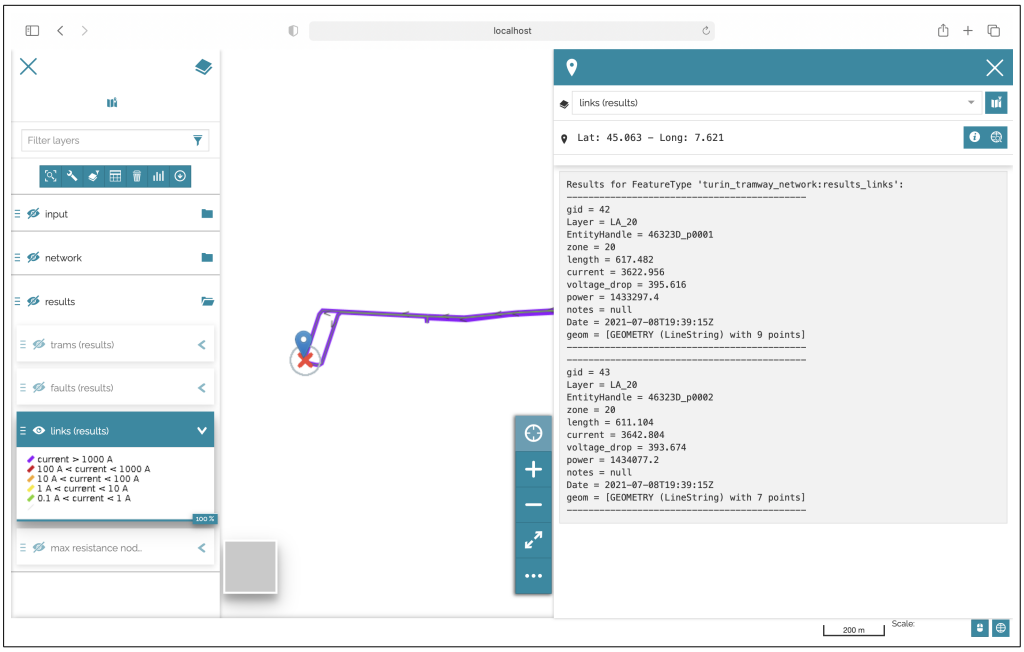

Figure 6.2: Visualisation of results on MapStore: details of two features of the `links` layer.

Figure 6.3 shows instead information on the `faults` layer: the current value, absorbed by the fault is indeed the sum of the two previous values, being:

$$I_f = I_1 + I_2 = 7{,}266\,\text{A} \tag{6.1}$$

Lastly, opening the attribute table of one of the layers allows to inspect all the features, which can be filtered and localised on the map by clicking on the lens at the left side of each row. The attribute table for the `links` layer is shown in Figure 6.4.

6.2 Validation with LTspice

To validate the results of the simulation described in the previous section, zone 20 is drawn on LTspice. The circuit diagram is shown in Figure 6.5. The resistance values are taken from the simulator, taking significant digits up to the microhm order of magnitude.

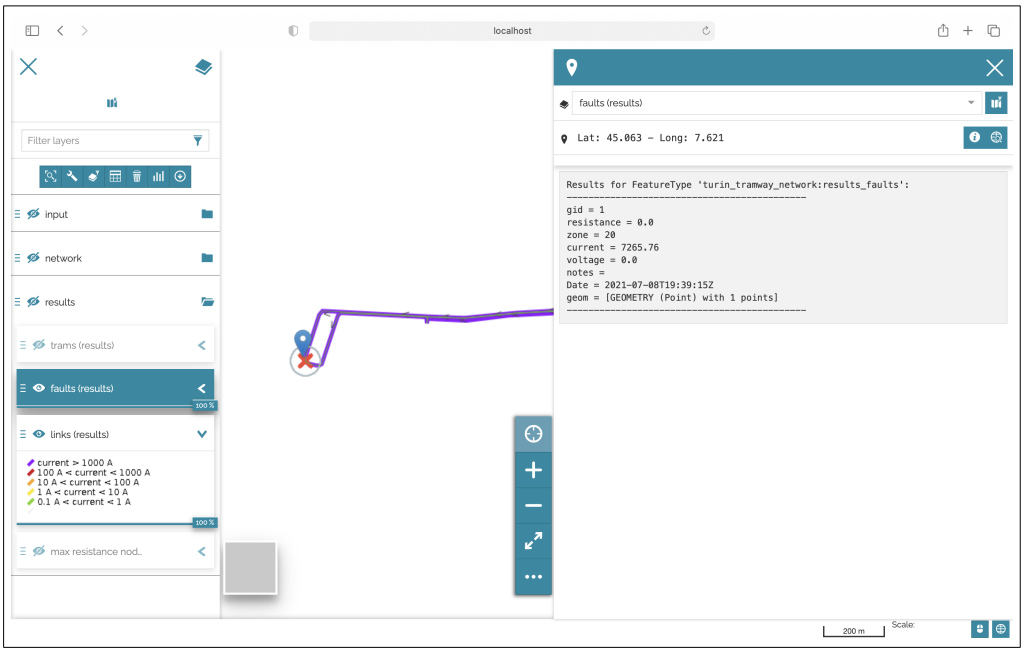

Figure 6.3: Visualisation of results on MapStore: detail of one feature of the `faults` layer.

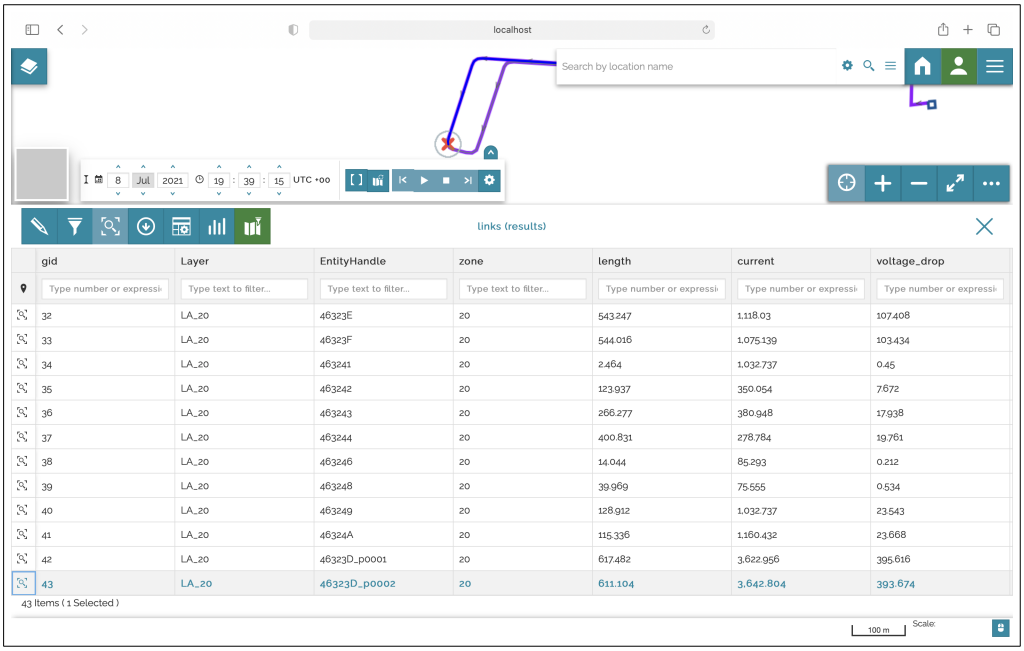

Figure 6.4: Visualisation of results on MapStore: attribute table of the `links` layer.

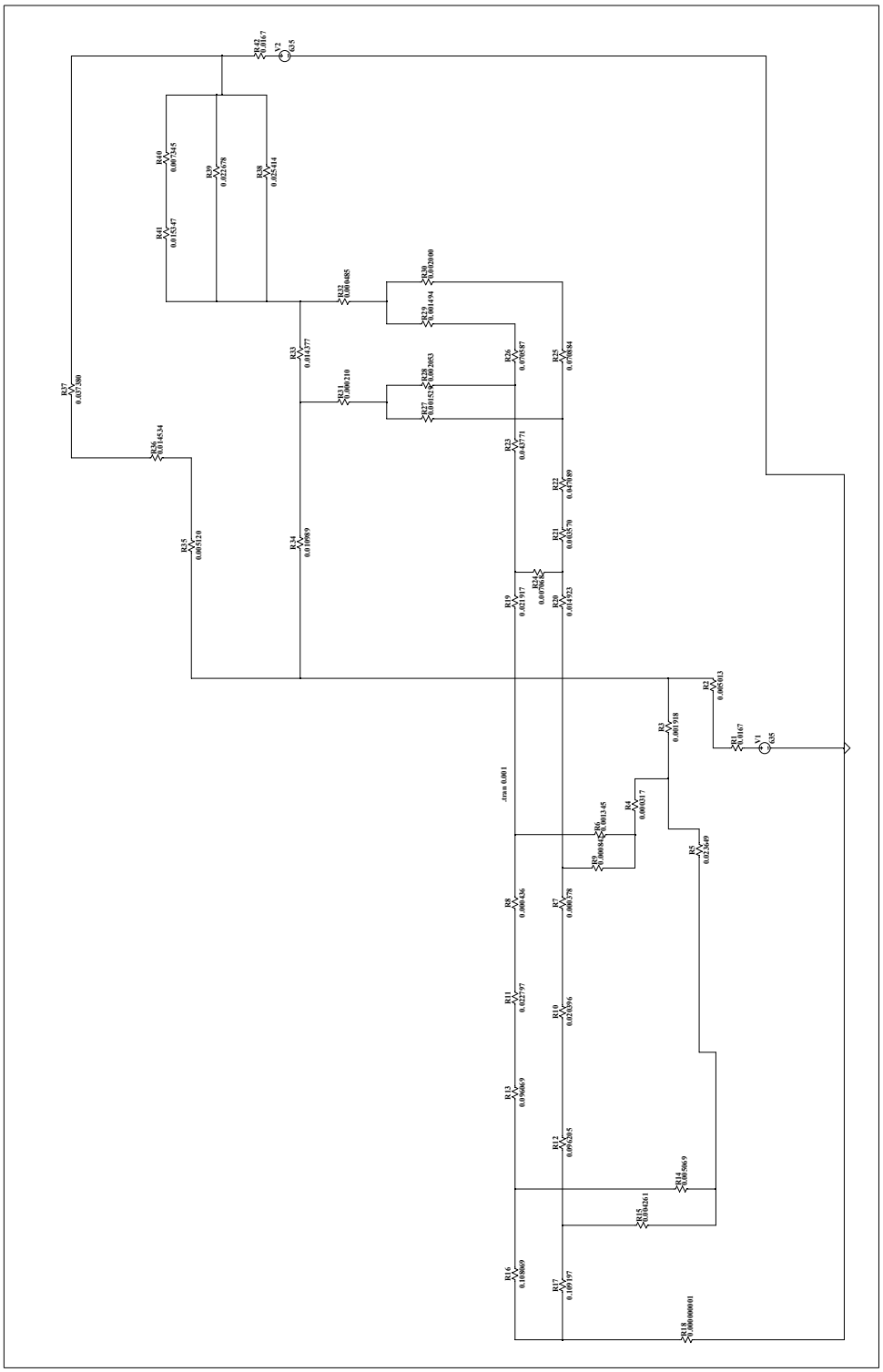

Figure 6.5: Circuit diagram of zone 20, on LTspice.

Three different simulations are carried out, for the different configurations of powering substations: in the first simulation, zone 20 is powered by the "Trapani" substation; in the second, it is powered by the "San Paolo" substation; in the third, both substations power the zone. The `tran` directive is used: it performs the transient analysis of the circuit. This requires to set a time interval for the analysis: 1 ms is chosen. Actually any small interval would be sufficient, since the circuit reaches instantly its steady state, not having any reactive element. The comparisons of the results between the simulator and the LTspice analysis for the three simulations are given respectively in Table 6.1, Table 6.2 and Table 6.3. Each one reports the values of current in the fault, as well as the currents exiting the two substations; plus, they report the voltages at all the junction points in the zone where feeder cables are connected to the OCS. Since junctions are not labelled in the simulator, the id of the OCS features that start at those points are used to identify these nodes.

ID		**Results**	
Simulator	**LTspice**	**Simulator**	**LTspice**
I_{fault}	I_{R18}	6,614 A	6,617 A
$I_{Trapani}$	I_{R1}	6,614 A	6,617 A
$I_{SanPaolo}$	I_{R42}	0 A	0 A
$V_{LA\_20-46322E}$	V_{N009}	479 V	479 V
$V_{LA\_20-46322F}$	V_{N010}	488 V	488 V
$V_{LA\_20-463231}$	V_{N027}	489 V	489 V
$V_{LA\_20-46322C}$	V_{N029}	370 V	370 V

Table 6.1: Simulation of zone 20 powered by the "Trapani" substation: comparison of the results.

6.3 Comparison with previous thesis work

As already mentioned, a MATLAB simulator for the Turin tramway electrification system had been developed in [1]; Chapter 5 of the work reported the results of four different simulations. The same simulations have been carried out on the new application and a comparison of the results is given.

ID		Results	
Simulator	**LTspice**	**Simulator**	**LTspice**
I_{fault}	I_{R18}	5,858 A	5,860 A
$I_{Trapani}$	I_{R1}	0 A	0 A
$I_{SanPaolo}$	I_{R42}	5,858 A	5,860 A
$V_{LA\_20-46322E}$	V_{N009}	424 V	424 V
$V_{LA\_20-46322F}$	V_{N010}	464 V	464 V
$V_{LA\_20-463231}$	V_{N027}	505 V	505 V
$V_{LA\_20-46322C}$	V_{N029}	328 V	328 V

Table 6.2: Simulation of zone 20 powered by the "San Paolo" substation: comparison of the results.

ID		Results	
Simulator	**LTspice**	**Simulator**	**LTspice**
I_{fault}	I_{R18}	7,266 A	7,269 A
$I_{Trapani}$	I_{R1}	4,430 A	4,432 A
$I_{SanPaolo}$	I_{R42}	2,836 A	2,838 A
$V_{LA\_20-46322E}$	V_{N009}	526 V	526 V
$V_{LA\_20-46322F}$	V_{N010}	552 V	552 V
$V_{LA\_20-463231}$	V_{N027}	572 V	572 V
$V_{LA\_20-46322C}$	V_{N029}	407 V	407 V

Table 6.3: Simulation of zone 20 powered by the "Trapani" and "San Paolo" substations in parallel: comparison of the results.

6.3.1 Zone 40, Duca d'Aosta, two trams

The first scenario includes two trams, located in zone 40, both of which absorb 1,000 A of current. The zone is powered by substation "Duca d'Aosta": its voltage source is set to $V_{cc} = 600$ V, with a series resistance $R_{ser} = 16.7\,\text{m}\Omega$. The MATLAB simulator of [1] reported voltages of 562 V and 547 V for the two trams. The results of the Python simulator developed in this thesis are shown in Figure 6.6 and Figure 6.7. The voltages of the two trams are respectively 562 V and 547 V, matching the others.

The simulation is repeated changing only the current absorption of the two trams to $-1{,}000$ A, which would represent injection of power into the network, due to regenerative braking. In this simulation, the power is then absorbed by the substations. It should be noted that it is not a feasible scenario, since the substations cannot absorb power: the only way regenerative braking could be exploited would be if other vehicles were present in the same zone and could absorb

the power released by the ones that are braking. Also in this case the results of the two simulators match, with voltages respectively of 638 V and 653 V.

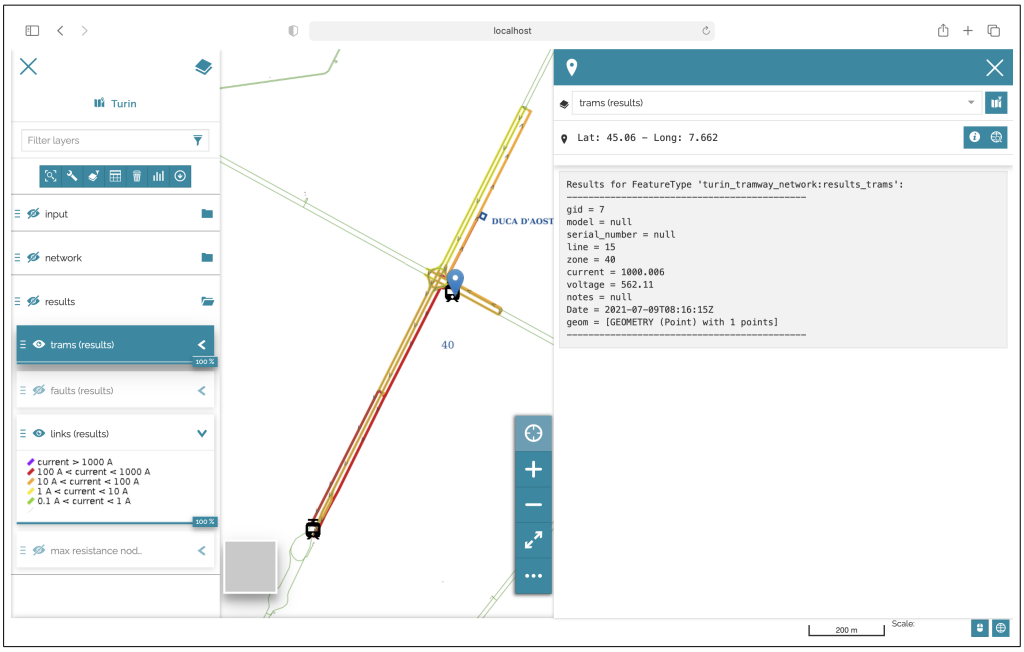

Figure 6.6: Simulation results of zone 40, powered by "Duca d'Aosta": voltage of the first tram.

6.3.2 Zone 40, Duca d'Aosta, one fault

Zone 40 is then simulated in presence of a fault, located almost where the second tram of the previous simulation was, right before the zone divider, which separates zone 40 from zone 30. First, substation "Duca d'Aosta" is considered active. In this simulator, the fault resistance was set to $R_f = 1\,\text{n}\Omega$, whereas in the previous work it was taken as a true short-circuit, with $R_f = 0\,\Omega$. As shown in Figure 6.8, the current through the fault is evaluated at $I_f = 17{,}520\,\text{A}$. [1] reported it at 17,474 A.

6.3.3 Zone 40, Sebastopoli, one fault

The same simulation is then repeated, replacing the substation "Duca d'Aosta" with the "Sebastopoli" one. The Python simulator gives $I_f = 16{,}066\,\text{A}$, while [1] reports $I_f = 14{,}934\,\text{A}$. The results are shown in Figure 6.9.

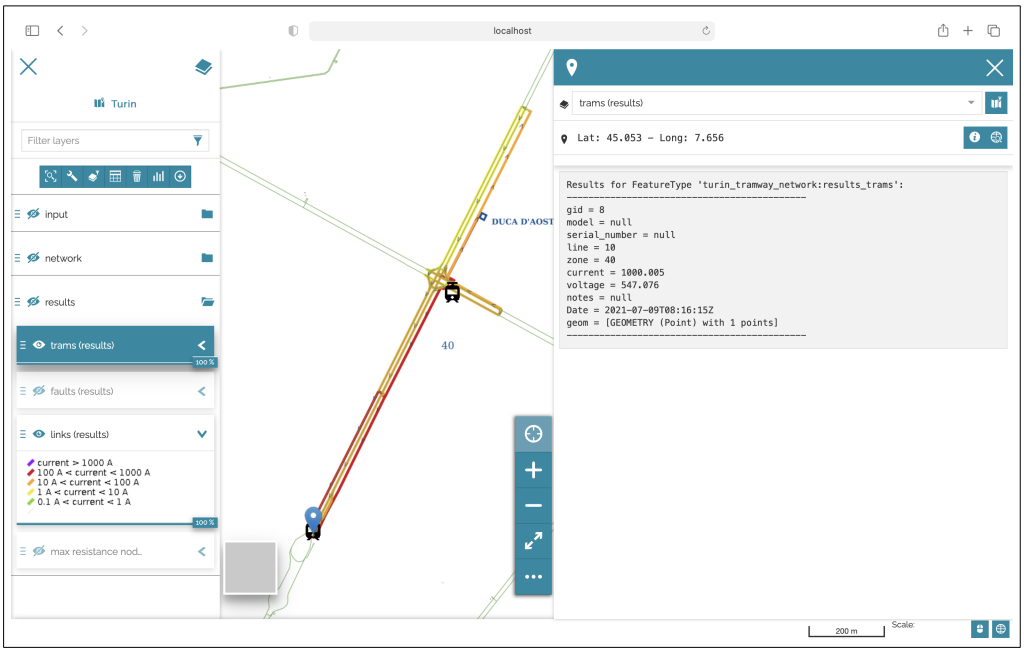

Figure 6.7: Simulation results of zone 40, powered by "Duca d'Aosta": voltage of the second tram.

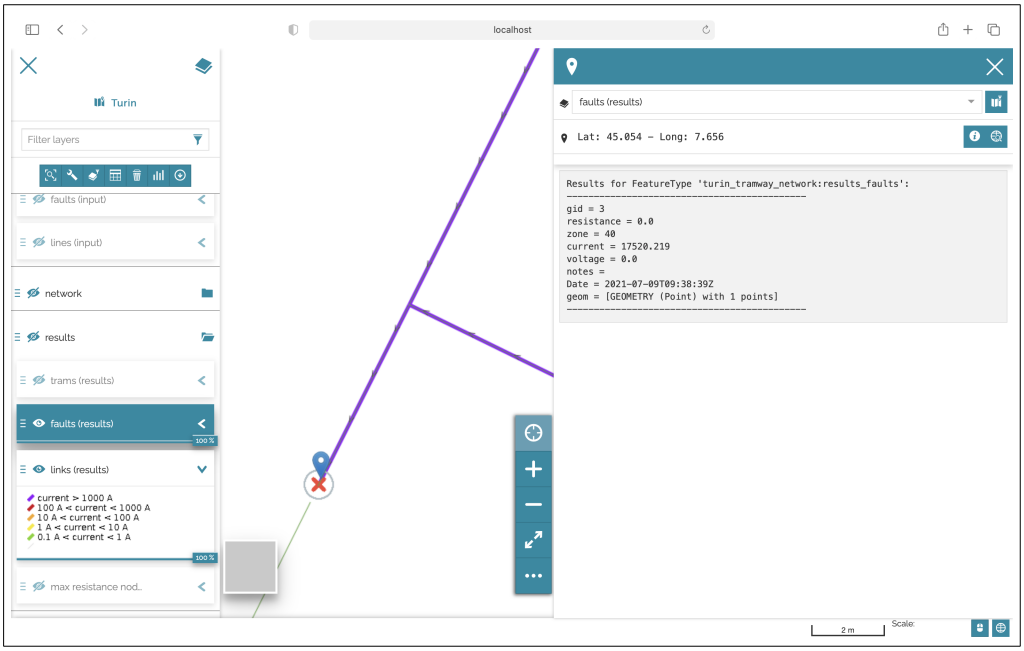

Figure 6.8: Simulation results of zone 40, powered by "Duca d'Aosta": current through the fault.

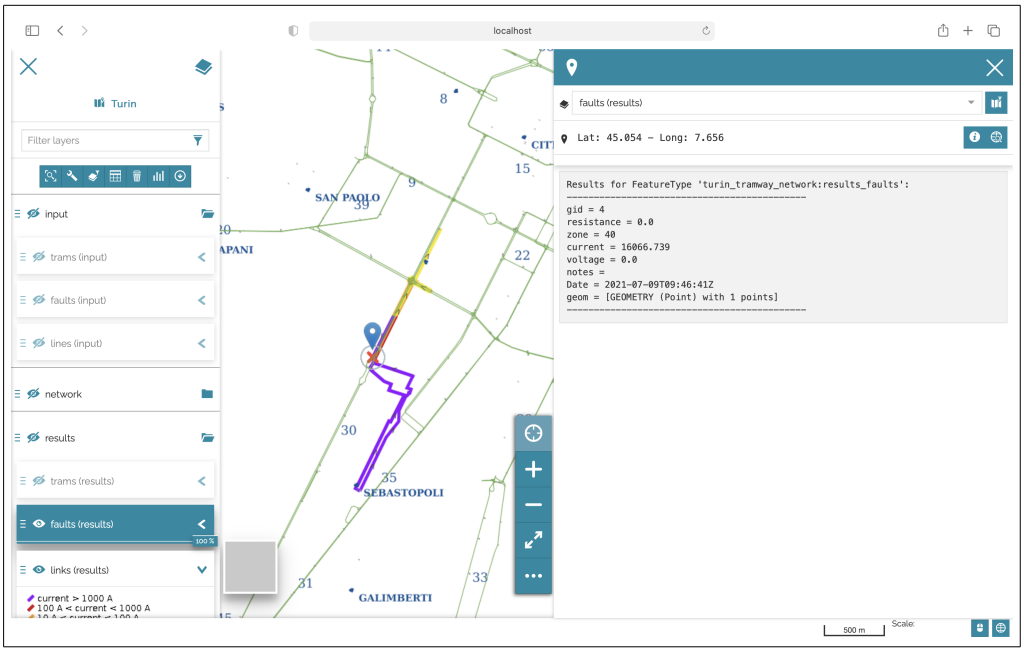

Figure 6.9: Simulation results of zone 40, powered by "Sebastopoli": current through the fault.

6.3.4 Zone 40, two substations in parallel, one fault

Finally, the simulation is repeated considering both the "Duca d'Aosta" and the "Sebastopoli" substations active, powering zone 40 in parallel. The results are: $I_f = 26{,}611\,\text{A}$ for the Python simulator, $I_f = 25{,}854\,\text{A}$ for the MATLAB one. The results are shown in Figure 6.10.

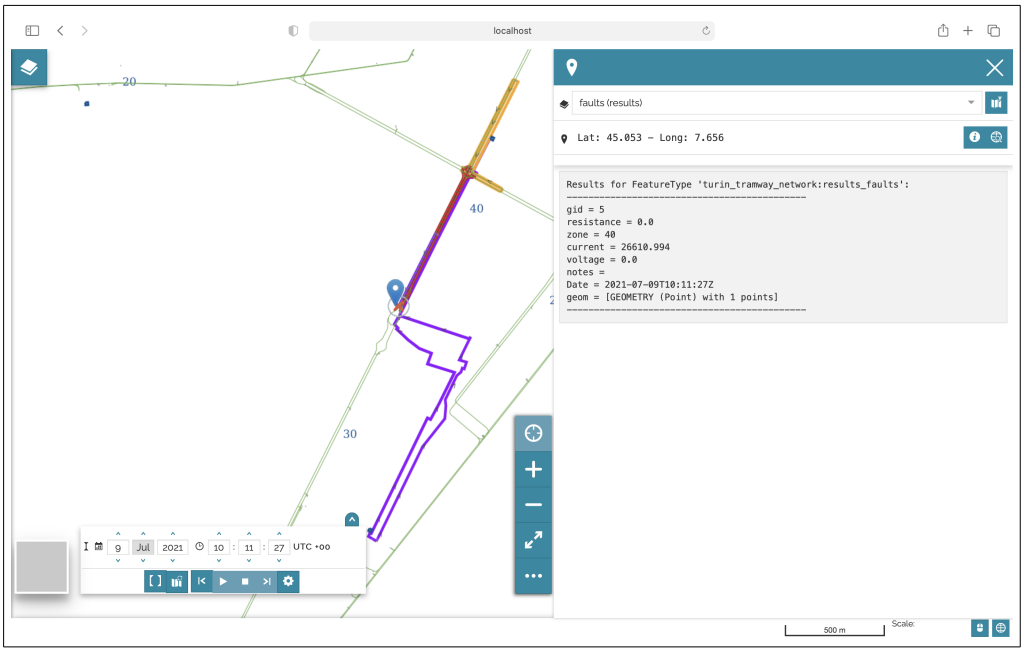

Figure 6.10: Simulation results of zone 40, powered by "Duca d'Aosta" and "Sebastopoli" in parallel: current through the fault.

Chapter 7

Conclusions

The proposed software tool satisfies the requirements set in Chapter 2, both in terms of the features of the simulator and in terms of the architecture of the whole application. The validation of the results performed with the LTspice simulator proves the correctness of the algorithms that build and solve the circuital model of the network. Additional tests should be carried out to verify the quality of the results of simulations that include tracks and negative cables in the model. To this end, given the difficulty of reproducing the complete network in a simulation environment, a comparison with some experimental data retrieved on the field could be the best approach.

As for possible improvements of the work, firstly the authentication and connectivity issues described in Chapter 4 should be addressed, in order to enhance the security and the flexibility of the system.

Moreover, as far as the simulator is concerned, a simple yet effective optimisation of the function that computes the point of maximum resistance for a given zone would be to first estimate the point without taking tracks into account and then redo the simulations including tracks, but testing just the nodes in a small neighbourhood of the previously estimated point, without having to iterate over the whole zone, a task that is very time consuming when tracks are included in the model. Furthermore, according to the need of addressing specific problems, the internal models of the trams, of the substations, and so forth, could be made more complex.

The way in which simulations results are outputted could be improved by developing a plugin that reports some text based messages, alerting the user in case of particular events, such as the occurrence of voltages or currents over given thresholds, or the presence of anomalies in the connections (disconnected cables, zones in parallel, etc.).

More in general, two considerations could still be made with respect to the applications of this platform: the first is that it could support quasi-real-time

simulations of the network, provided that real-time data of location and current absorption of the vehicles are available. This could greatly enhance the level of load monitoring on the different network zones. The second is that the proposed GIS platform could be used to integrate both other data sources, in the forms of layers, and other applications, in the form of MapStore plugins, following the approach already used in this thesis.

Bibliography

[1] Lorenzo Bertolone Citin. «Procedura automatica per il calcolo delle correnti di cortocircuito e dei flussi di potenza nella rete tranviaria torinese = Automatic procedure for calculating the short circuit currents and the power flows in the Turin tramway network». Luglio 2018. URL: http://webthesis.biblio.polito.it/7666/ (cit. on pp. 1, 3, 52, 57–59).

[2] Enrico Pons, Riccardo Tommasini, and Pietro Colella. «Fault Current Detection and Dangerous Voltages in DC Urban Rail Traction Systems». In: *IEEE Transactions on Industry Applications* 53.4 (2017), pp. 4109–4115. DOI: 10.1109/TIA.2017.2692202 (cit. on pp. 1, 2).

[3] T.K. Ho, Baohua Mao, Z.Z. Yuan, H.D. Liu, and Yu-Fai Fung. «Computer simulation and modeling in railway applications». In: *Computer Physics Communications* 143(1) (Feb. 2002), pp. 1–10. DOI: 10.1016/S0010-4655(01)00410-6 (cit. on pp. 4, 5).

[4] Mehmet Söylemez and Süleyman Açıkbaş. «Multi-train Simulation of DC Rail Traction Power Systems with Regenerative Braking». In: vol. 15. May 2004 (cit. on p. 6).

[5] *OpenPowerNet*. URL: https://www.openpowernet.de (visited on 07/12/2021) (cit. on p. 7).

[6] *Rail Power Systems - eTraX™*. URL: https://etap.com/solutions/railways (visited on 07/12/2021) (cit. on p. 7).

[7] *Consulting and planning for rail electrification*. URL: https://www.mobility.siemens.com/global/en/portfolio/rail/electrification/planning-and-consulting.html (visited on 07/12/2021) (cit. on p. 7).

[8] *TracFeed products*. URL: https://www.tracfeed-produkte.de/en/ (visited on 07/12/2021) (cit. on p. 7).

[9] *Signon - Portfolio*. URL: https://en.signon-group.com/en/products (visited on 07/12/2021) (cit. on p. 7).

[10] *Fabel*. URL: https://www.enotrac.com/en/software-tools/fabel.php (visited on 07/12/2021) (cit. on p. 7).

[11] Anders Nyman. «TTS/SIMON Power Log - A Simulation Tool For Evaluating Electrical Train Power Supply Systems». In: *WIT Transactions on The Built Environment* 37 (1998), pp. 427–436. DOI: 10.2495/CR980411 (cit. on p. 7).

[12] Lars Abrahamsson. «Railway Power Supply Models and Methods for Long-term Investment Analysis». PhD thesis. Sept. 2008. DOI: 10.13140/RG.2.1.1258.7040 (cit. on pp. 7, 8).

[13] Alekhya Datta and Parimita Mohanty. «Enterprise GIS and Smart Electric Grid for India's power sector». In: *2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*. 2013, pp. 1–7. DOI: 10.1109/ISGT.2013.6497806 (cit. on p. 9).

[14] P. A. Parikh and T. D. Nielsen. «Transforming traditional geographic information system to support smart distribution systems». In: *2009 IEEE/PES Power Systems Conference and Exposition*. 2009, pp. 1–4. DOI: 10.1109/PSCE.2009.4839979 (cit. on p. 10).

[15] Najmeh Rezaei, Majid Nayeripour, A Roosta, and Taher Niknam. «Role of GIS in Distribution Power Systems». In: *World Academy of Science, Engineering and Technology* 36 (Dec. 2009) (cit. on p. 10).

[16] M.V. Krishna Rao, B.S. Varma, and C. Radhakrishna. «Experiences on implementation of GIS based tools for analysis, planning and design of distribution systems». In: *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*. 2008, pp. 1–8. DOI: 10.1109/PES.2008.4596690 (cit. on p. 10).

[17] R. Santodomingo, Eduardo Pilo, J.A. Mondejar, and M. García-Vaquero. «Adapting the CIM model to describe electrified railway systems». In: Aug. 2008, pp. 381–390. ISBN: 9781845641269. DOI: 10.2495/CR080381 (cit. on p. 11).

[18] Xianqi Li, Xiaoliang Feng, Zhiyuan Zeng, Xuejun Xu, and Yongchuan Zhang. «Distribution feeder one-line diagrams automatic generation from geographic diagrams based on GIS». In: *2008 Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies*. 2008, pp. 2228–2232. DOI: 10.1109/DRPT.2008.4523781 (cit. on p. 11).

[19] Yuxing Duan, Chengyou Wang, and Wenjun Zhou. «Topology modeling of distribution network based on open-source GIS». In: *2011 4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT)*. 2011, pp. 527–530. DOI: 10.1109/DRPT.2011.5993948 (cit. on p. 11).

[20] Sundara Bharathi Dhamotharan. *GIS based web application for Electricity Asset Management System*. July 2015. DOI: 10.13140/RG.2.1.1811.1843 (cit. on p. 11).

[21] R.T. Fielding and R.N. Taylor. «Principled design of the modern Web architecture». In: *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*. 2000, pp. 407–416. DOI: 10.1145/337180.337228 (cit. on p. 14).

[22] *Web Services Architecture, W3C Working Group Note 11 February 2004*. URL: https://www.w3.org/TR/ws-arch/ (visited on 06/09/2021) (cit. on p. 14).

[23] David Moretz. «Internet GIS». In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar and Hui Xiong. Boston, MA: Springer US, 2008, pp. 591–596. ISBN: 978-0-387-35973-1. DOI: 10.1007/978-0-387-35973-1_648. URL: https://doi.org/10.1007/978-0-387-35973-1_648 (cit. on p. 14).

[24] Christopher D. Michaelis and Daniel P. Ames. «Web Feature Service (WFS) and Web Map Service (WMS)». In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar, Hui Xiong, and Xun Zhou. Cham: Springer International Publishing, 2017, pp. 2485–2488. ISBN: 978-3-319-17885-1. DOI: 10.1007/978-3-319-17885-1_1480. URL: https://doi.org/10.1007/978-3-319-17885-1_1480 (cit. on p. 15).

[25] *OpenGIS Web Feature Service 2.0 Interface Standard*. Open Geospatial Consortium Inc. URL: https://www.ogc.org/standards/wfs (cit. on p. 15).

[26] Abhinaya Sinha. «Web Feature Service (WFS)». In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar, Hui Xiong, and Xun Zhou. Cham: Springer International Publishing, 2017, pp. 2481–2485. ISBN: 978-3-319-17885-1. DOI: 10.1007/978-3-319-17885-1_1479. URL: https://doi.org/10.1007/978-3-319-17885-1_1479 (cit. on p. 15).

[27] *Apache Tomcat*. URL: http://tomcat.apache.org (visited on 07/11/2021) (cit. on p. 16).

[28] *GeoServer*. URL: http://geoserver.org (visited on 07/11/2021) (cit. on p. 16).

[29] *PostGIS*. URL: http://postgis.net/ (visited on 07/11/2021) (cit. on p. 16).

[30] *MapStore*. URL: https://mapstore.readthedocs.io/en/latest/ (visited on 07/11/2021) (cit. on p. 16).

[31] *MapStore - Main Frontend Technologies*. URL: https://mapstore.readthedocs.io/en/latest/developer-guide/reactjs-and-redux-introduction.html (visited on 06/08/2021) (cit. on p. 17).

[32] *Python*. URL: https://www.python.org (visited on 07/11/2021) (cit. on p. 17).

[33] *Docker*. URL: https://docs.docker.com/get-started/overview/ (visited on 06/08/2021) (cit. on p. 18).

[34] *MapStore Developer Guide - MapStore2 users GeoServer integration with Authkey*. URL: https://mapstore.readthedocs.io/en/latest/developer-guide/integrations/users/geoserver/ (visited on 07/07/2021) (cit. on p. 31).

[35] *GeoServer User Manual*. URL: https://docs.geoserver.org/master/en/user/index.html (visited on 06/24/2021) (cit. on p. 31).

[36] *MapStore User Guide*. URL: https://mapstore.readthedocs.io/en/latest/user-guide/home-page/ (visited on 06/08/2021) (cit. on p. 33).

[37] *MapStore Developer Guide - Database Setup*. URL: https://mapstore.readthedocs.io/en/latest/developer-guide/database-setup/ (visited on 07/07/2021) (cit. on p. 40).