Hochschule Neubrandenburg
University of Applied Sciences

University of Applied Sciences Neubrandenburg

Geoinformatics

Development of an Information System for the Management of GNSS Station Metadata using GeodesyML

Bachelor Thesis

submitted by: Johannes Karl Kindermann

To Obtain the Academic Degree of

"Bachelor of Engineering" (B.Eng.)

First Supervisor: Prof. Dr.-Ing. Andreas Wehrenpfennig

Second Supervirsor: M.Eng. Markus Bradke

Submitted on 10.08.2022

URN: **urn:nbn:de:gbv:519-thesis-2022-0262-2**

Abstract

GNSS station metadata needs to be machine-readable due to changing demands of the user segment and need for interoperability with other systems. We propose an information system implemented as a web application for the management of GNSS station metadata that is both interoperable and highly usable. It implements the GeodesyML standard for exchanging geodetic metadata. The system is designed to be easily extensible and accommodate diverse use cases. It outperforms comparable systems in key user experience metrics.

Acknowledgements

I would like to thank M.Eng. Markus Bradke and Prof. Dr.-Ing. Andreas Wehrenpfennig for guidance during the making of this thesis. I am especially grateful for having had the opportunity to work at GFZ Potsdam as an intern from July to December 2021. I would like to thank my family and friends for being supportive during my studies.

Contents

Chapter 1

Introduction

Global navigation satellite systems (GNSS) play an essential role in many parts of society. Since the first GNSS was made publicly available in 1995, this role has changed significantly. While originally most applications of GNSS were in science or geodesy, over 60% of the total revenue generated by GNSS over the next 10 years is estimated to come from consumer applications and over 29% from road and automotive applications [1]. The increasing revenue from consumer applications is a continued trend and also reflected in the revenue of added-value services which is expected to grow by 11% annually [1].

To accommodate these use cases, GNSS data has to be findable, accessible, interoperable and reusable, such that it can be processed by automated systems. For each of these properties, metadata is a key factor. An important piece of metadata in GNSS is the *sitelog*, describing in detail a configuration of a GNSS receiver and GNSS antenna. Sitelogs were introduced in the early days of GNSS and, as a purely human-readable file format, do not facilitate automated processing.

A proposed standard called GeodesyML is designed to solve this problem but lacks broader adoption as few implementations exist. To increase adoption, there needs to be a system that includes full support for GeodesyML but also includes all of the features in current systems that manage GNSS station metadata with ASCII sitelogs. This way, station operators can be motivated to switch station metadata to the GeodesyML format, which improves the infrastructure of important GNSS networks. This thesis is concerned with the development of such a system.

Chapter 2

Global Navigation Satellite Systems

Global Navigation Satellite Systems (GNSS) are satellite constellations that provide positioning, navigation and timing (PNT) services. As a large chunk of the data generated in total is georeferenced, GNSS are an integral part of modern society. Applications of GNSS range from personal navigation systems to high-precision surveying, precise timing and a multitude of scientific applications.

2.1 Operation

The method underlying positioning with GNSS is the simultaneous measurement of the range of four or more GNSS satellites to a GNSS receiver. These each determine the radius of a sphere around a particular satellite, on whose surface the receiver is located. An intersection of four or more spheres yields a singular point, thereby uniquely identifying the position of the receiver in space.

There are several methods for measuring range to a satellite. In each case, the physical observable is a radio signal sent by the satellite. These radio signals have multiple components: carrier wave, ranging code and navigation data. The carrier wave is a harmonic in the L band, which ranges from 1 to 2 GHz in frequency. The ranging code is a periodic binary sequence of pseudo-random noise and the navigation data is a binary sequence encoding useful information about the satellite.

The first of these methods is measuring pseudorange. This is done by creating a replica of a signal's ranging code in the receiver and comparing it to the signal's ranging code, thereby determining the time offset of the signal, its travel time and ultimately range to the satellite. The precision of this method is on the order of $10^{-1}\mathrm{m}$.

The second method is measuring the phase of the carrier wave of the incoming signal. This can be used to count the number of peaks in the carrier wave, which also determine range.

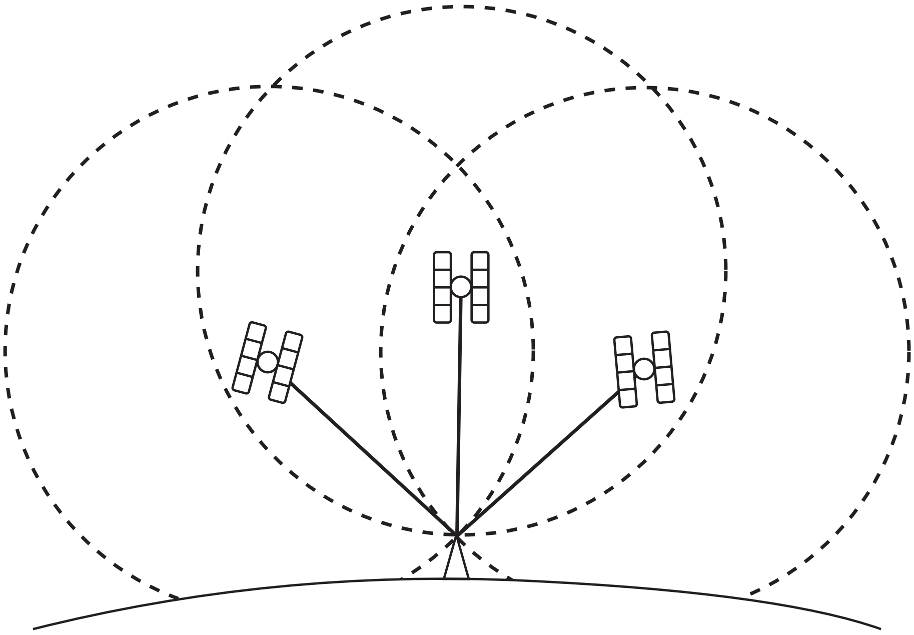


Figure 2.1: Method of positioning using GNSS under idealized conditions.

Precision of this method is on the order of $10^{-3}\mathrm{m}$, but it is unstable in case of signal loss.

Both methods rely on the synchronization of the clocks of the satellite and receiver. Since this is not given in practice, timing errors must be considered in the measurement of ranges. The timing error of the satellite is broadcast to the receiver as part of a signal's navigation message. However, the timing error of the receiver adds an additional unknown quantity to the range measurements. Assuming perfect conditions, measurements of three range quantities would suffice to position the receiver. To account for the receiver timing error, an additional fourth range must be measured.

For strategic reasons, there are multiple GNSS constellations. The major constellations are GPS (USA), GLONASS (RF), Galileo (EU) and BeiDou (China). Each includes 20 to 40 satellites to always provide line of sight to at least the minimum number of satellites required for PNT at any point on the Earth's surface.

Each GNSS is made up of three segments: the space segment, control segment and user segment. The space segment comprises the constellation of satellites. The control segment is responsible for maintaining the health of the system by monitoring broadcast signals and uploading navigation data to satellites. It consists of a group of monitoring stations, ground antennas and a master control station dispersed in the region covered by the GNSS. The user segment consists of any other GNSS receivers used for both civil and military purposes.

2.2 The International GNSS Service

The International GNSS Service (IGS) is a worldwide organisation of self-funding agencies, universities and research institutions, such as the GFZ German Research Centre for Geosciences, that pool the data of their permanent GNSS stations[1]. The IGS uses this data to provide products such as GNSS satellite ephemerides and clocks, coordinates of its constituent stations, earth rotation parameters and atmospheric parameters. It also provides GNSS data for its stations. In addition to precise positioning, the data distributed by the IGS sees use in scientific applications such as the realization of the International Terrestrial Reference Frame, monitoring the deformation of the solid earth, monitoring sea-level change and climate change events as well as producing ionospheric and tropospheric maps [2]. The IGS to date receives data from 512 stations from 350 member organisations.

At the highest level, it is comprised of the Governing Board making policy decisions and the Central Bureau providing day-to-day management. The Governing Board maintains several committees, such as the Infrastructure Committee. Data Centers provide access to IGS data and products, where data is directly gathered from IGS Tracking Stations, and products are provided by Analysis Centers operating on the data. In addition, the Governing Board maintains several Pilot Projects and Working Groups. The Infrastructure Committee furthers the development of GeodesyML in an internal task team.

2.3 GNSS Station Metadata

A GNSS station is described by its metadata. This is information such as how the station can be identified, where it is located and what kind of hardware was installed at a specific time. Besides being essential for the overall maintainability of a GNSS network, it is also used in the post-processing of GNSS data. Post-processing is the method of improving raw observations using data unknown at the time of measurement. Significant errors in GNSS measurements include satellite clock and ephemeris error, atmospheric error, receiver noise and multipath. Changes of receiver equipment and possible multipath effects are documented in a station's metadata alongside many other relevant data points. Having up-to-date knowledge of a station's metadata allows accurate post-processing and avoiding discontinuities in the time series of a station's measurements.

[1] A GNSS station is a GNSS receiver coupled to a GNSS antenna, alongside other hardware that is optional such as meteorological sensors.

Chapter 3

GNSS Station Sitelogs

A document describing the metadata of a GNSS station is called a sitelog. The de facto standard for this purpose is a file format based entirely on human-readable text, the *ASCII* sitelog. Originally introduced by the IGS for managing the metadata of its own stations, it has been in use for more than 20 years [3]. Since there are disadvantages to a purely human-readable approach, especially in terms of automating management of station metadata, there has been an effort to introduce a machine-readable[1] file format for the same purpose called *GeodesyML*.

3.1 ASCII

When the IGS was founded in 1994 and GPS as the first GNSS became fully operational in 1995, machine-readable formats were not available or not considered for GNSS station metadata. The Extensible Markup Language (XML) file format for example was started in 1996. In addition, adoption of a binary format could have hindered contributions to IGS due to lack of software and/or hardware on the part of station operators. This way, a format was chosen similar to the Receiver Independent Exchange Format (RINEX) [6], then available in its second version and based on ASCII. The chosen format was subsequently adopted by other GNSS networks such as the EUREF Permanent Network (EPN). The IGS publishes a template [5] and a set of instructions [4] for filling out a sitelog.

Initially, these documents were managed by hand, involving network administrators in maintaining the metadata of every station in a network. Later, most organizations administering GNSS networks transitioned to using web-based systems where station operators are able to update metadata without other involvement. These include the IGS Site Log Manager [7], EPN M3G [8], GFZ German Research Centre for Geoscience semisys [9] and Geoscience

[1] Machine-readable taking to describe a file format that is unambiguously readable. An ASCII sitelog *can* be read and parsed by a computer, but for lack of validation not without ambiguity.

#	Name	Description
0.	Form	Information on the current document: author, date, etc.
1.	Site Identification of the GNSS Monument	Schemes by which station can be distinguished: four-character ID, IERS DOMES number, etc. Description of the monument including geology.
2.	Site Location Information	City, state, country, tectonic plate and ITRF coordinates
3.	GNSS Receiver Information	History of installed GNSS receivers
4.	GNSS Antenna Information	History of installed GNSS antennas and radomes
5.	Surveyed Local Ties	Geodetic measurements to stations of other geodetic techniques, such as SLR, VLBI, etc.
6.	Frequency Standard	Historical and current clocks references by the receiver, either internal or external. If external, options such as a hydrogen maser or caesium atomic clock apply.
7.	Collocation Information	Other geodetic techniques at this location
8.	Meteorological Instrumentation	History of installed meteorological sensors: humidity, pressure, temperature, water vapour or other. As a station's measurements are influenced by tropospheric weather conditions, knowledge of these parameters is useful.
9.	Local Ongoing Conditions Possibly Affecting Computed Position	History of interferences, multipath sources and signal obstructions
10.	Local Episodic Effects Possibly Affecting Data Quality	Diverse events such as equipment malfunction or accumulation of snow
11.	On-Site, Point of Contact Agency Information	Organisation/department and people in charge of physically managing station
12.	Responsible Agency	Agency responsible for metadata, if different to previous section
13.	More Information	Information such as data center and URLs for further inquiry. ASCII picture of antenna and radome including measurements.

Table 3.1: Sections of an ASCII sitelog [4] [5]

Australia GNSS Site Manager [10].

In connection to automated processing of ASCII sitelogs by such systems several issues have become apparent with the file format. Since there is no mechanism to unambiguously validate an ASCII sitelog, station operators often deviate from the intended format when entering data. This poses a problem for automated systems relying on correctness and complicates the already difficult process of parsing ASCII sitelogs. In addition, the standard as maintained by the IGS is unversioned, so systems are unable to tell if an imported document conforms to the latest standard [3].

In its 2021+ Strategic Plan [11], the IGS has advocated for standardization and interoperability in multi-GNSS data and products. This includes GNSS station metadata and reference frameworks such as the FAIR principles (findability, accessibility, interoperability and reuse of digital assets) [12]. Making GNSS station metadata in its ASCII sitelog form adhere to these standards would require significant changes of the format, such as adding licensing [13] [14].

3.2 GeodesyML

To address the shortcomings of the ASCII sitelog format, in 2016 the Australian and New Zealand geodetic agencies proposed a new standard called GeodesyML[2] with support by the former[3] IGS Data Center Working Group [15]. It is an application schema of the Geography Markup Language (GML) itself implemented using the Extensible Markup Language (XML).

XML is a file format whose main purpose is serialization, i.e. to serve as an intermediary format between two systems that have different internal representations of data. XML is also a markup language and structures information in *tags* or *elements*. As this structure can be defined by a strict grammar called a *schema*, an XML document can be *validated* against that schema to test whether it is well-formed. This allows to avoid ambiguity in parsing XML documents, making them machine-readable. Since XML is a markup language, it is also human-readable.

GML, as defined in the ISO 19136 standard, is an XML implementation of ISO 19107 *Spatial schema*. It is designed for modeling, transport and storage of geographic data [16]. It provides a rich vocabulary of primitive elements for representing geometry, coordinates, reference systems, time and others that can be used to create *application schemas* for a particular problem domain such as geodesy. It deals with features, being the fundamental unit of geographic information [16]. An application schema only uses a subset of the vocabulary available in GML and combines it with additional details from the problem domain.

[2] https://github.com/GeoscienceAustralia/GeodesyML

[3] Merged into the Infrastructure Committee.

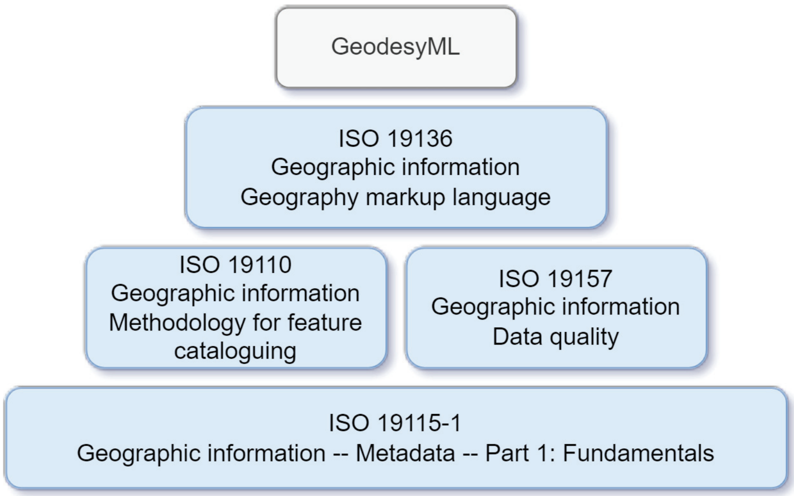


Figure 3.1: GeodesyML builds on other standards, c.f. [17].

GeodesyML solves the problem of geodetic data and metadata often not being interoperable, as is the case with ASCII sitelogs. Interoperability is defined by three principles [16]:

- The ability to find what you need when you need it;
- Once located, the ability to access and obtain what is needed;
- And after obtaining it, to be able to understand it and put it to good use.

Standards play a major role in providing interoperability. As such, GeodesyML builds on standards that enable these principles.

Besides GML, important additional standards that GeodesyML builds on top of are ISO ISO19115-1, ISO 19110 and ISO 19157. ISO 19115-1 *Geographic information – Metadata – Part 1: Fundamentals* is foundational to the format. It provides many different elements for describing the metadata of geographic datasets such as title, responsible party, reference system, file identifier and point of contact [16]. This metadata is a prerequisite for reuse and aids especially in locating required data.

ISO 19110 *Geographic information – Methodology for feature cataloguing* and ISO 19157 *Geographic information – Data quality* provide more specific information about features and the quality of data than covered in ISO 191115-1 [17]. This way, they help to better obtain, understand and use data.

GNSS station metadata is only a subset of GeodesyML. It also accommodates geodetic data like positions, measurements and reference frames. Within the subset of station metadata, GeodesyML stores additional information compared to ASCII sitelogs that also increases interoperability. See 3.2.

In contrast to ASCII sitelogs, GeodesyML follows a strict versioning scheme. It is currently in version 0.5 and is seeing development by the IGS Infrastructure Committee with proposals

Element	Description
siteContacts	List of responsible parties for a station.
siteDataCenters	List of data centers from which GNSS data by a station may be obtained.
siteDataSource	A responsible party from which GNSS data by a station may be obtained.
DOI	Digital Object Identifier
dataStreams	Specification of a URL from which data may be obtained in a specific format and sampling interval. Also stores NTRIP mounts. Relevant for RTK.

Table 3.2: Some of the additional information stored in a GeodesyML sitelog compared to an ASCII sitelog. For a comparison between ASCII and GeodesyML sitelogs for the same station, see systems that implement exports in both standards, such as [8].

such as [13]. With continued development, it is on track to be endorsed as an IGS standard. In the future, GeodesyML may also be adopted by other geodetic techniques such as SLR, VLBI and DORIS, as these use sitelogs similar to the GNSS ASCII sitelog. GeodesyML is general enough to also accommodate these other techniques.

Chapter 4

Analysis

Systems analysis is an important part of the software development lifecycle in which facts are gathered and requirements for a system are derived from these facts. The success of software development projects is strongly linked with the quality of the work done at this stage.

4.1 User Stories

A user story is an informal, natural language description of features of a software system, written from the perspective of an end user. An example of this could be: ”As a guest, I can register, so that I can log in.” A user story is similar to a use case, but more general. Use cases and functional requirements can be derived from user stories.

User stories can be organised in user story maps. A user story map is a grid diagram of several user stories. The horizontal axis defines a narrative through the application that the user may follow to achieve her goals. The vertical axis describes the generality of a user story. The main business activities of users, being the most general, are aligned at the top. Below follow the main narrative and more specific user stories that may depend on stories above. User stories further down the vertical axis may be assigned to later iterations of the product.

In the diagrams presented here, the color of a note indicates its swimlane with the exception of yellow being used to indicate external responsibility. If notes are stuck together, one is dependent on the other. An arrow indicates an additional dependency.

For the current system, the stories of guests and regular users, station operators, network administrators and administrators were considered in user story maps. The Tasks swimlane contains those user stories to be completed for the minimum viable product (MVP); the Subtasks swimlane those for later iterations. The MVP is the minimally feature complete version of a software to be used in a production environment.

Guests or regular users as shown in figure 4.1 can view public data. This includes public

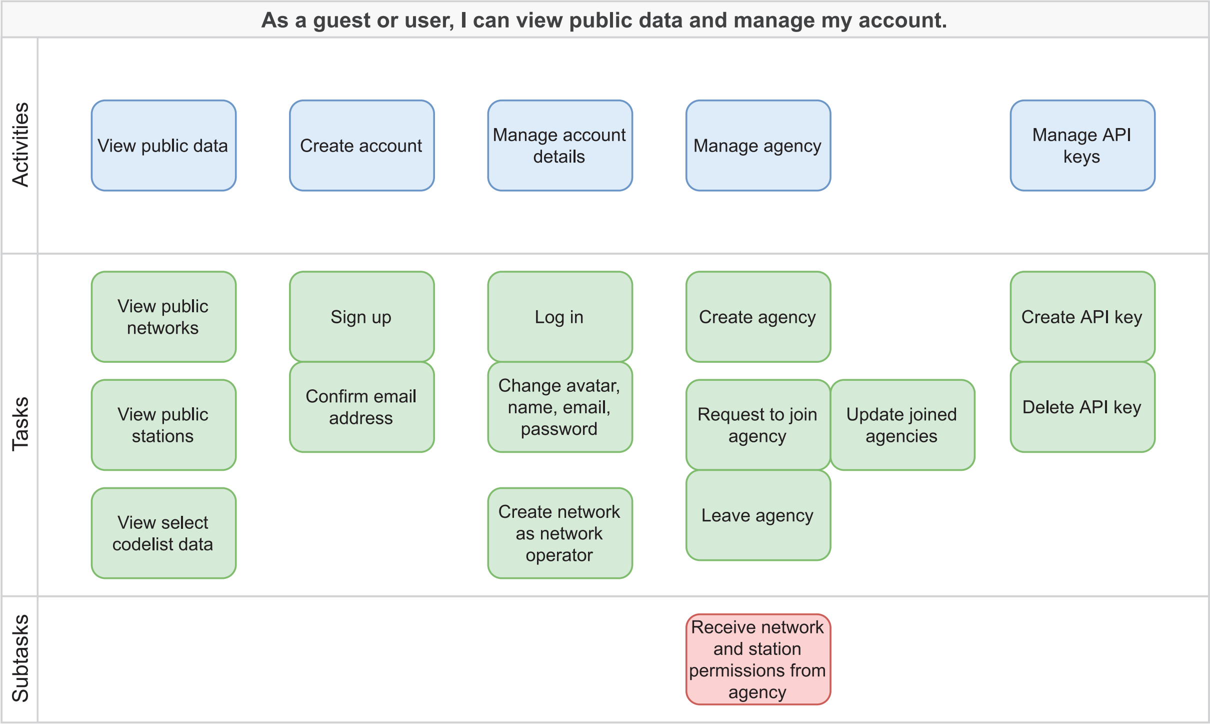


Figure 4.1: User story map for guests and regular users.

networks, stations and codelist data. In addition to permitting assignment of permissions on networks and stations via roles, both can be set to be generally public or generally private. This is less relevant for public networks such as IGS but more relevant for networks not publicly visible such as SAPOS. Codelist data[1] are sets of values that are useful in the representation of a station log. Examples include antenna and receiver hardware models, tectonic plates, countries, etc.

A guest may create an account. Any created account must be validated by email and by an administrator. Once completed, the user can manage her account details, agencies and API keys. Agency membership is used to determine whether a user can change the details of an agency, affecting station logs referencing that agency. In the future, agencies may also be used to assign permissions on networks or stations automatically. The user can also create a network and will be assigned network operator role for it on creation.

Station operators (SO) as shown in figure 4.2 are more privileged than regular users and can manage stations and associated station logs. A station log is a format-independent abstraction of a sitelog. The SO role includes the abilities to view, update, delete and manage the assignment of roles for a station. The SO user story also includes main business objectives 2, 4 and 5 from the network administrator user story on a station level.

A SO can set the digital object identifier (DOI) and license of a station to be inherited by logs for which these values are unset. These fields are required for reasons outlined in e.g. [13].

[1] Represented in GeodesyML by the CT_CodelistCatalogue element.

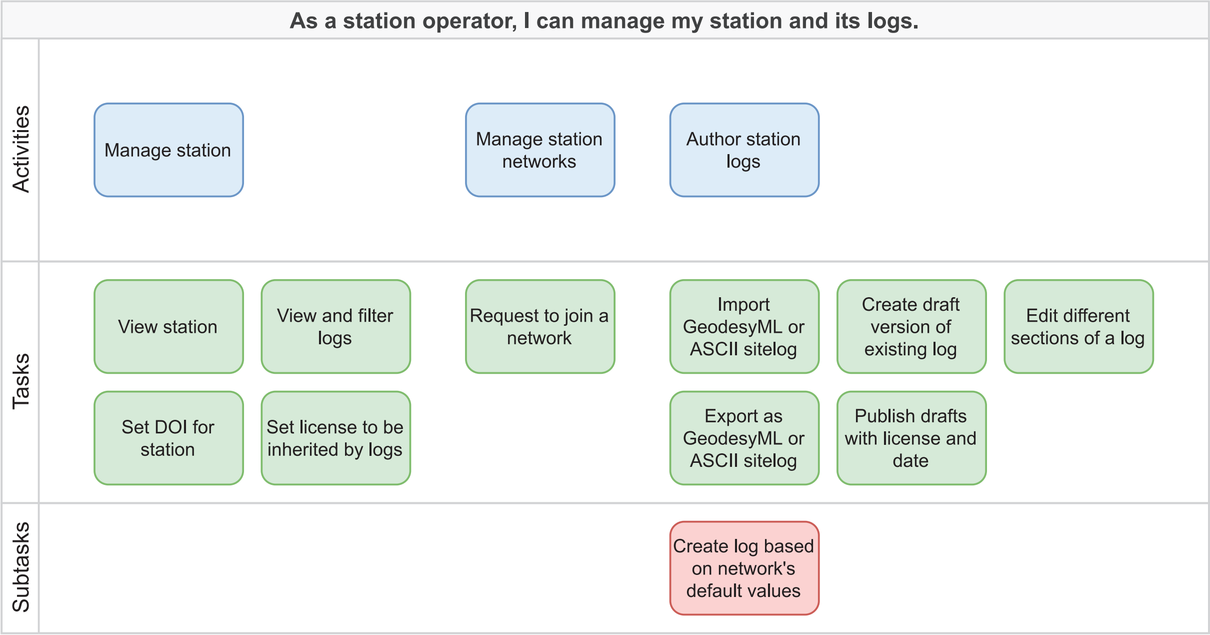

As a station operator, I can manage my station and its logs.
Activities
Tasks
Subtasks
Manage station
Manage station networks
Author station logs
View station
View and filter logs
Request to join a network
Import GeodesyML or ASCII sitelog
Create draft version of existing log
Edit different sections of a log
Set DOI for station
Set license to be inherited by logs
Export as GeodesyML or ASCII sitelog
Publish drafts with license and date
Create log based on network's default values

Figure 4.2: User story map for the station operator role.

The SO can also request for a station to join a network, but does not have the ability to set the master network of a station. Every station can only belong to a single master network. The master network of a station is used to assign permissions to users authorized with the network on every member station.

The ability to edit a station also allows the SO to edit the logs of the station, including importing, exporting, editing, drafting and publishing of logs. As the primary use case of the system is to update existing station logs, creation of ”empty” logs based on default values specified by the master network is not part of the MVP. See figures 9.2 and 9.3 for a detailed description of the process of drafting, updating and publishing a station log.

Network administrators (NA) as shown in figure 4.4 are responsible for managing a network and its stations. NAs can create, view and delete stations in a network. For stations that are part of a master network on which a user has the NA role, the user also has SO access. In future iterations, an NA may request to assign a different network as the master network of a station and for a station of the network to join another network.

An important part of the NA’s role is to control user access to the network. This is achieved by creating and assigning roles with abilities from the set of ”view”, ”update”, ”delete” and ”manage users”. In addition to the ”network administrator” role of a network having every permission, there could e.g. also be an ”auditor” role only having the ”view” permission. To assign a role to a user, the user is invited to take up that role. The invited user may accept or decline the invitation and it may also be revoked by a NA. See figure 9.1 for a detailed description of the invitation process. Furthermore the NA may set the default visibility of the network to be either public or private. When set to public, the ”view” permission of every role on the network is ignored but not removed.

To better facilitate multi-network use, NAs are able to define criteria which the logs of stations are to be validated against. This is important because different real-world networks have different metadata standards. The IGS network for example requires all receiver hardware listed in its station logs to be present in a predefined set of values considered valid[2]. The GFZ network and others impose no such restrictions and also consider unregistered receiver models to be valid. To aid in validation, each network is assigned an empty validation set on creation to which an NA may assign existing validation. The NA may also select to inherit from one or more existing validation sets, on top of which the validation set of the inheriting network will be applied. This way, the strict validation set of the IGS network could be inherited by another network and refined with additional rules instead of having to be rebuilt. The creation of entirely new validation rules and the ability to manually set errors and warnings for a field without an associated rule are not implemented as part of the MVP.

It is essential for NAs to stay up-to-date on the infrastructure of the network. This is achieved with notifications in the system's notification centre and with emails to the user's address. Should a station, i.e. its current log, fail validation, both channels are used to inform NAs. In addition, NAs receive a system notification every time there are changes in a network station or its logs. An email digest with configurable frequency and contents for such events is outside of the scope of the MVP.

To make the system interoperable, i.e. work with both other instances of the same software as well as other systems such as those in 5, NAs should be able to automatically synchronize the changes made in one system with other systems to avoid having to perform changes multiple times and risking input errors. To authenticate with other systems, NAs must be able to store API keys of those systems. The ability to push updates to other systems is planned for a later iteration, whereas the ability for other systems to pull updates from the current system is the responsibility of the respective systems.

Administrators (AD) as shown in figure 4.3 manage the instance of the software (”system”), its users, codelists and settings. Having control over other users and the ability to confirm signups and requests of users to join an agency, it carries high responsibility and should be assigned accordingly. Creating, editing and deleting of codelist values is exclusive to ADs. They also choose when to generate GeodesyML codelist documents from these values to be referenced in GeodesyML exports of station logs. An instance provides configurable settings manageable by ADs for emails, API access and syndication feeds such as RSS or Atom. In

[2] https://files.igs.org/pub/station/general/rcvr_ant.tab

future iterations, this will include settings for gathering GDPR consent from users via email (relevant to systems with large user bases), publishing syndication feeds for private networks and stations, and configuring when and how to prune outdated station logs.

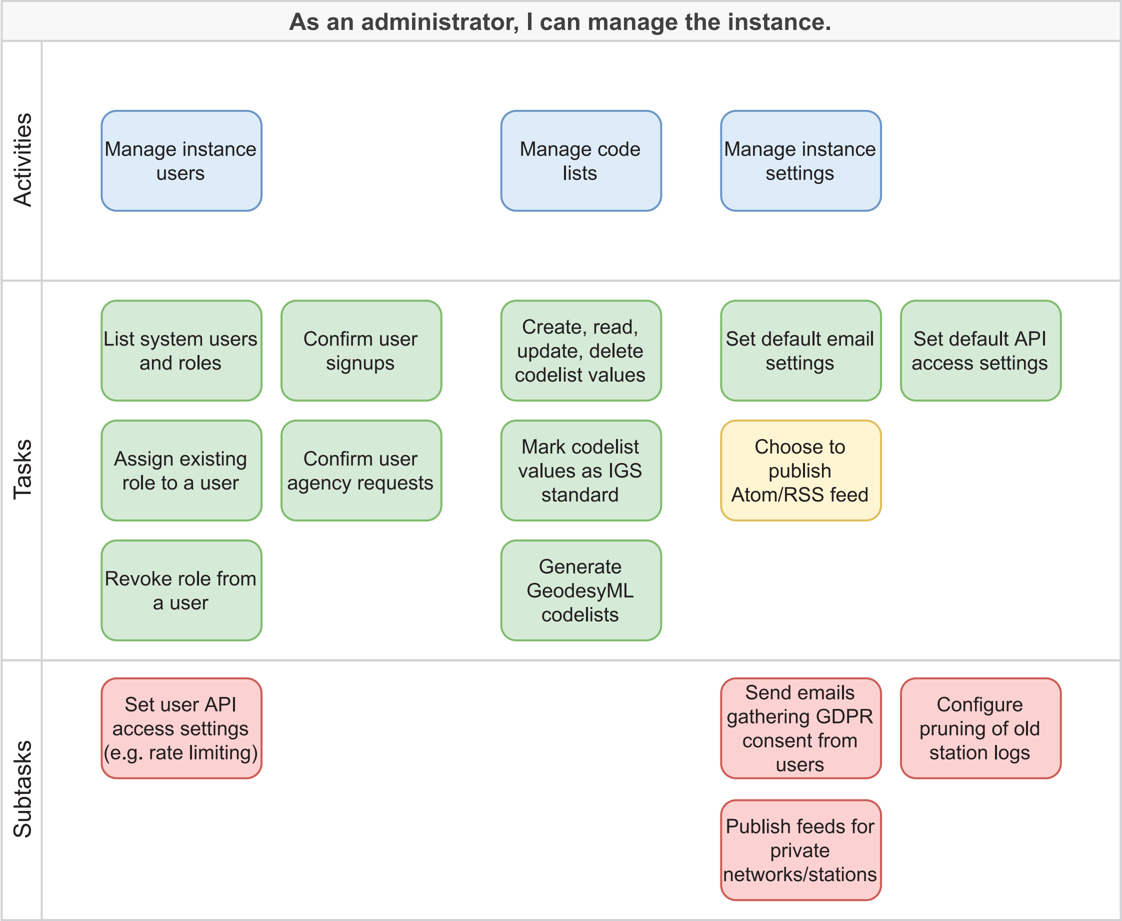

As an administrator, I can manage the instance.
Activities
Manage instance users
Manage code lists
Manage instance settings
Tasks
List system users and roles
Confirm user signups
Create, read, update, delete codelist values
Set default email settings
Set default API access settings
Assign existing role to a user
Confirm user agency requests
Mark codelist values as IGS standard
Choose to publish Atom/RSS feed
Revoke role from a user
Generate GeodesyML codelists
Subtasks
Set user API access settings (e.g. rate limiting)
Send emails gathering GDPR consent from users
Configure pruning of old station logs
Publish feeds for private networks/stations

Figure 4.3: User story map for the administrator role.

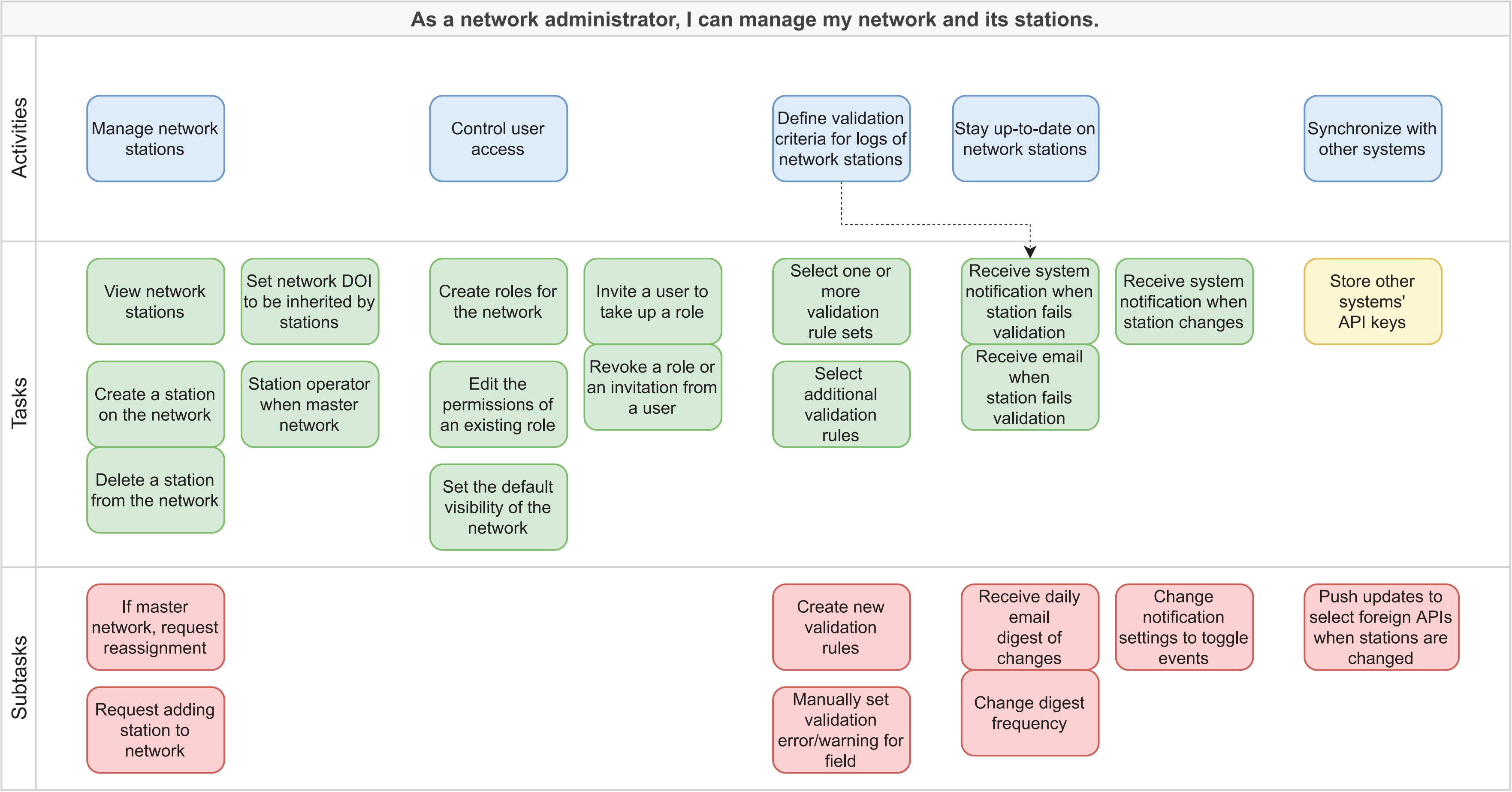

As a network administrator, I can manage my network and its stations.
Activities
Tasks
Subtasks
Manage network stations
Control user access
Define validation criteria for logs of network stations
Stay up-to-date on network stations
Synchronize with other systems
View network stations
Set network DOI to be inherited by stations
Create roles for the network
Invite a user to take up a role
Select one or more validation rule sets
Receive system notification when station fails validation
Receive system notification when station changes
Store other systems' API keys
Create a station on the network
Station operator when master network
Edit the permissions of an existing role
Revoke a role or an invitation from a user
Select additional validation rules
Receive email when station fails validation
Delete a station from the network
Set the default visibility of the network
If master network, request reassignment
Create new validation rules
Receive daily email digest of changes
Change notification settings to toggle events
Push updates to select foreign APIs when stations are changed
Request adding station to network
Manually set validation error/warning for field
Change digest frequency

Figure 4.4: User story map for the network administrator role.

4.2 Functional Requirements

Functional requirements can be derived from the user stories and corresponding use cases. Functional requirements are those that specify the behaviour of a system, as opposed to non-functional requirements that specify aspects of the operation of the system in general.

Access to the system is provided to users both via a graphical user interface (GUI) as well as an application programming interface (API) in the form of representational state transfer (REST). In the GUI, users access the system's data using list, map and tile/card views.

Responsiveness is the feature of a GUI to fluidly adjust to changes in screen layout. A responsive web application has similar user experience on mobile, desktop and other devices. The GUI of the system is responsive and delivers high user experience on a mobile device. A station operator may want to update her station's metadata in the field, requiring a responsive user interface.

Authentication is the act of verifying the identity of a user. The system provides mechanisms for authentication in both GUI and API. Users are authenticated in the API using Bearer tokens (”keys”). Optionally, the system integrates with existing on-premise authentication schemes such as LDAP.

Authorization is the act of verifying that a given user is privileged to execute an action. Users are authorized based on group membership in a granular way. Groups include guests, administrators, network administrators and station operators. Groups can be assigned by administrators, network administrators or station operators on their respective level of privilege.

Storage of data input into the system is realized in a relational database management system (RDBMS) equipped with a geospatial extension. Such an extension allows management and analysis of geographic data, effectively turning the RDBMS into a geographic information system. The database is able to store multiple versions of metadata for a station at the same time. This is used for the creation of sitelog drafts. Additionally, the database is replicated to a hot standby server as a failover mechanism.

Validation is performed on any sitelog before it is input into and output from the system. If a sitelog contains invalid formatting, it is not stored. Validation of sitelogs ensures that no invalid metadata is ever published for a station. Validation rules are arbitrary in complexity and can be defined globally or for the stations of a particular network. Examples of validation rules for a particular network are the requirement for IGS stations to be assigned a valid DOMES number and to be equipped with GNSS receiver and antenna hardware recognized as valid by the IGS. An example of a global validation rule is the requirement for receiver equipment to be registered with correct installation dates, i.e. no GNSS receiver is ever registered as installed before a previous receiver is removed. Validation rules are subdivided into rules that cause errors and rules that cause warnings. If a sitelog is validated with errors, it is not able to be published, but is able to be published if validated with warnings only.

Updates are sent to other systems, such as those outlined in 5, when relevant changes are made to the system's data. Since standardization of the APIs of similar systems cannot be expected, they will have to be wrapped by a unified interface. This feature can also be implemented by a component external to the core system by using its REST API. Optionally, updates can be received from other systems in the reverse case.

The **API** lets applications access a subset of the system's features. This includes retrieving and updating station logs, retrieving station pictures, retrieving information about receivers and antennas including mean and individual phase center offset for receiver antennas, retrieving metadata about specific networks and retrieving data about countries. The API endpoints are documented using the OpenAPI specification and integration-tested.

The **GUI** contains views for different user stories: user view, station operator view, network manager view and administrator view.

SEO is provided by adding correct HTML meta tags and a site map, enabling crawlers to navigate the site more efficiently. In addition, the web site has high scores (greater than 90) in Lighthouse criteria performance, accessibility, best practices and SEO. The Core Web Vitals are especially important for this purpose.

4.2.1 Inputs

Inputs to the system are accepted from various data sources. Most importantly, IGS sitelogs both in ASCII and in GeodesyML form are able to be input into the system. In addition, the system accepts inputs of the following data:

IGS rcvr_ant.tab, antenna.gra contain valid GNSS receiver and GNSS antenna models as accepted by the IGS and their possible configurations.

ISO 3166 defines codes for the representation of names of countries and their subdivisions.

ANTEX (Antenna Exchange Format) files contain calibrations for GNSS satellite and receiver antennas. Only data on receiver antennas is relevant to this system. *Optional*

Ocean loading describes a displacement of sea-adjacent land mass by the elastic response of the earth's crust to ocean tides, thereby influencing the position of GNSS stations. Data for the correction of this effect is available. *Optional*

Site pictures

Logs of station visits to potentially record immediately any changes made to a station during a visit. *Optional*

Tectonic plates data based on the NNR-MORVEL56 [18] model.

Networks data. This includes networks with public access such as IGS as well as networks with private access such as SAPOS.

4.3 Non-Functional Requirements

Non-functional requirements, also called *quality requirements*, specify aspects of the operation of the system in general.

Reliability and availability are provided by database replication. A hot standby database server runs in parallel with the main server and is switched into operation should that server fail. This ensures there is minimal downtime. In addition, backups of the database are performed daily. The source code of the application is hosted on a different server in an instance of the version control software Git, and deployed to the application server.

Extensibility describes both the ability to extend a system as well as to refactor existing code in a system. Extensibility is given by adherence to standard practices during development and by providing developer documentation. There is no way to add plugins to the system or extend it at runtime.

Browser Independence is required as users and station operators are spread across the globe. Internationalization and localization are provided in all but translations. The system is not translated to a language other than English. This is similar to comparable systems, as English is highly prevalent in the field.

Scalability and Efficiency are required to handle a steadily growing number of stations to be stored in the system. The database schema must hence be efficient and based on the needs of access via the GUI and API.

Usability is key to ensuring user retention. The web application should be easy and self-explanatory to use. Having a straightforward visual hierarchy ensures that the application is easy to understand. Accessibility criteria such as sufficient contrast between content and background or screen reader labels for unlabeled buttons benefit users in general. In case a part of the application has potential to be confusing and cannot be changed, help is provided.

4.4 Technical Requirements

The PHP-based web framework Laravel[3] is used to implement the API. The software utilizes open-source software components and the source code is eventually released under a permissive license. The service is accessible from anywhere in the world.

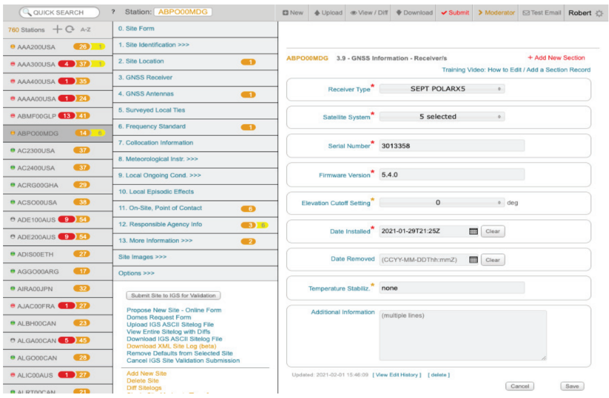
[3] https://laravel.com/

Chapter 5

Existing Systems

Different web-based systems exist for the management of GNSS station metadata. The capabilities of every system are according to the maintaining organizations' needs and there is no standard or "one size fits all".

5.1 International GNSS Service



ABPO00MDG 3.9 - GNSS Information - Receiver/s

Receiver Type* SEPT POLARX5

Satellite System* 5 selected

Serial Number* 3013358

Firmware Version* 5.4.0

Elevation Cutoff Setting* 0 deg

Date Installed* 2021-01-29T21:25Z

Date Removed (CCYY-MM-DDThh:mmZ)

Temperature Stabiliz.* none

Additional Information (multiple lines)

Updated: 2021-02-01 15:46:09 [View Edit History] [delete]

Figure 5.1: SLM user interface for viewing a sitelog.

The International GNSS Service provides the Site Log Manager (SLM) service at [11]. It was designed to assist station operators of stations contributing to the IGS with updating ASCII sitelogs of these stations.

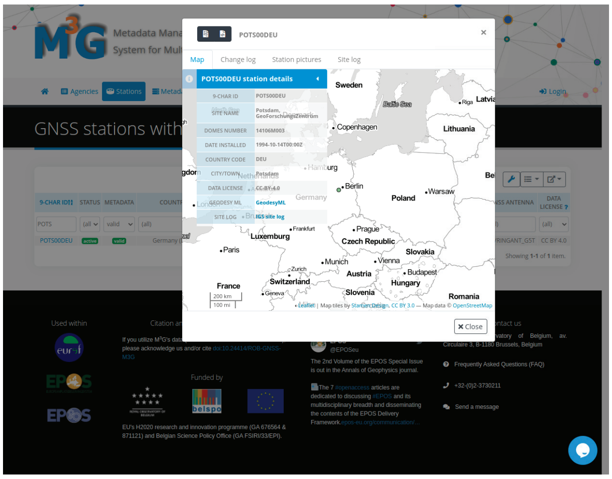
It systematically parses uploaded ASCII sitelogs, validating them in the process. As a result of validation, sitelog fields are marked as "error" when there is a violation of sitelog syntax and as "preferred" when a field is empty that does not constitute an error. Station operators are then prompted to fix these errors/warnings without involvement of the IGS network administrator[1]. SLM further allows station operators to save partial information and submit their changes on confirmation.

[1] The name of the position at the IGS is Network Coordinator.

Station metadata is saved in a MySQL database. From this database, other products relevant to the IGS are generated, such as SINEX files, lists of the network's stations and equipment and pictures of network stations. SLM however does not facilitate access to metadata as the service itself is access-restricted and metadata is instead published via FTP. To date, 148 agencies have registered with SLM and metadata of more than 750 active, former and proposed stations is stored in the system.

Since the system was originally built to target version 5.0 of the PHP programming language, which stopped receiving support on 01.01.2019, a new system *SLM 2.0* is being built for the same purpose but utilizing an updated technology stack, consisting of MySQL, Python and the Python framework Django. It is set to enter operation in 2022. The source code of the system will be made publicly available under an open-source license. It is designed to be extensible, so that other network administrators can deploy the system for their own purposes. In addition to feature parity with the previous system, it will include a REST API and support for GeodesyML sitelogs.

5.2 EUREF Permanent Network



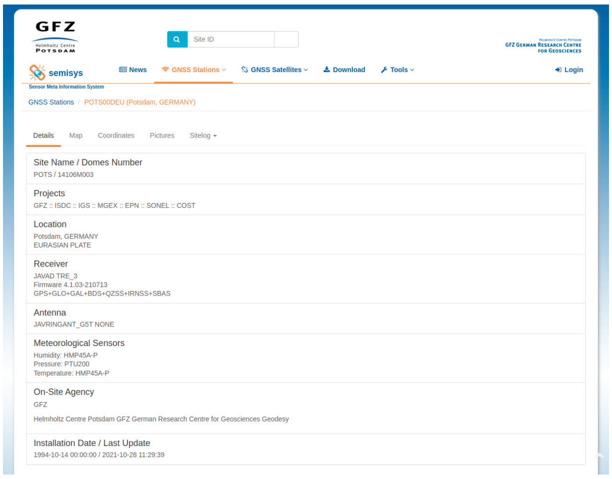
POTS00DEU station details

9-CHAR ID	POTS00DEU
SITE NAME	Potsdam, GeoForschungsZentrum
DOMES NUMBER	14106M003
DATE INSTALLED	1994-10-14T00:00Z
COUNTRY CODE	DEU
CITY/TOWN	Potsdam
DATA LICENSE	CC-BY-4.0
GEODESY ML	GeodesyML
SITE LOG	IGS site log

Figure 5.2: M3G user interface for viewing a sitelog.

Similar to the IGS, the EUREF Permanent Network (EPN) is a network of permanently operating GNSS stations. It was created in 1995 with the primary goal of supporting and improving the European Terrestrial Reference System (ETRS89) and successive realizations. Metadata for stations that are part of EPN is maintained in the Metadata Management and distribution system for Multiple GNSS Networks (M3G) at [8]. M3G is developed and maintained by the Royal Observatory of Belgium. To date, it contains metadata on over 4916 stations and in addition to EPN also manages other networks such as EPOS and EPN densification which are closely aligned with EPN.

5.3 GFZ German Research Centre for Geosciences


GFZ
Helmholtz Centre POTSDAM
Site ID
Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences
semisys
Sensor Meta Information System
News
GNSS Stations
GNSS Satellites
Download
Tools
Login
GNSS Stations / POTS00DEU (Potsdam, GERMANY)
Details
Map
Coordinates
Pictures
Sitelog

Site Name / Domes Number
POTS / 14106M003

Projects
GFZ :: ISDC :: IGS :: MGEX :: EPN :: SONEL :: COST

Location
Potsdam, GERMANY
EURASIAN PLATE

Receiver
JAVAD TRE_3
Firmware 4.1.03-210713
GPS+GLO+GAL+BDS+QZSS+IRNSS+SBAS

Antenna
JAVRINGANT_G5T NONE

Meteorological Sensors
Humidity: HMP45A-P
Pressure: PTU200
Temperature: HMP45A-P

On-Site Agency
GFZ

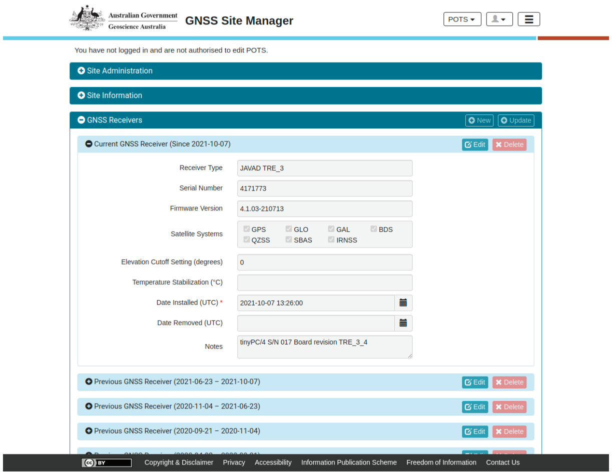
Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences Geodesy

Installation Date / Last Update
1994-10-14 00:00:00 / 2021-10-28 11:29:39

Figure 5.3: semisys user interface for viewing a sitelog.

The GFZ German Research Centre for Geosciences maintains a network of 55 permanent stations throughout the world. It is an important contributor to the IGS with 24 of those stations. The Operational Data Center group at GFZ also processes positioning data from over 800 additional stations in the IGS, EPN and SAPOS networks. As such, it maintains the Sensor Meta Information System (semisys) [9] for managing metadata. The system was initially developed in 2012 [19] and offers similar features to the IGS SLM. Data of the application is stored in a PostgreSQL database. The web interface was built using PHP and jQuery.

5.4 Geoscience Australia



You have not logged in and are not authorised to edit POTS.

Current GNSS Receiver (Since 2021-10-07)

Receiver Type	JAVAD TRE_3
Serial Number	4171773
Firmware Version	4.1.03-210713
Satellite Systems	☑ GPS ☑ GLO ☑ GAL ☑ BDS ☑ QZSS ☑ SBAS ☑ IRNSS
Elevation Cutoff Setting (degrees)	0
Temperature Stabilization (°C)	
Date Installed (UTC) *	2021-10-07 13:26:00
Date Removed (UTC)	
Notes	tinyPC/4 S/N 017 Board revision TRE_3_4

Previous GNSS Receiver (2021-06-23 – 2021-10-07)

Previous GNSS Receiver (2020-11-04 – 2021-06-23)

Previous GNSS Receiver (2020-09-21 – 2020-11-04)

Figure 5.4: GNSS Site Manager user interface for viewing a sitelog.

Geoscience Australia (GA) is an agency of the Australian government and likewise an important contributor to the IGS with 36 stations. It maintains approximately 1500 stations in several networks across the Australian region and South Pacific. In contrast to other systems, the solution used by GA [10] is open-source[2]. The web interface was built using the Angular framework.

5.5 Comparison

Table 5.1: Comparison of existing systems for GNSS station metadata management.

	SLM	SLM 2.0[1]	M3G	semisys	GSM	Proposed[1]
Version Control	◐	◐	◐	●	○	●
GeodesyML Import	○	○[2]	○	○	●	●
GeodesyML Export	○	●[2]	●	○	●	●
Multi-Network	○	○	●	●	◐	●
Validation	○	●	●	○	○	●
Open-Source	○	●	○	○	●	●

● Fully implemented
◐ Partially implemented
○ Not implemented

The systems can be evaluated by different criteria and contrasted with the system proposed here. A system is considered to partially implement version control if it allows saving of unpublished changes. It fully implements version control if several versions of a station log can be stored simultaneously. Version control is closely related to drafting. Systems are considered to support multi-network use cases if different networks can reside in the same system with different station assignments, validation rules and user permissions. A system implements validation if more fine-grained checking than required and preferred is supported.

[2] https://github.com/GeoscienceAustralia/GNSS-Site-Manager

[2] In development

[2] Not part of MVP

Chapter 6

Design

6.1 System Architecture

The system as defined by the requirements follows a multitier architecture. A multitier architecture is most commonly made up of physically separated presentation, application and a data tiers. Each tier is a black box that communicates with adjacent tiers via well defined interfaces. The system also makes use of the client-server model, in which an entity called a server provides services and resources that are requested by a client.

6.1.1 Data Tier

The data tier wraps mechanisms to access and persistent the data of the system in an API accessible to the application tier. Ideally, this is done without creating dependencies on the internal structure of the data tier. In the current system, the data tier can be realized by a single database management system, interfacing with the application tier via SQL. This tier can be expanded if a specific caching solution such as Redis is required in the future.

As per the functional requirements of the system, a relational database management system is used for data storage. Per the technical requirements, focus is placed on open source software components. There are many popular open-source RDBMSs available. PostgreSQL with the PostGIS extension was chosen. It holds the second-highest market share among open-source RDBMSs [20].

6.1.2 Application Tier

The application tier runs the business logic of the system and moves data inbetween the presentation and data tiers. As per the technical requirements, the Laravel PHP framework is used on a PHP application server such as Apache. It interfaces with the presentation tier via a REST

API.

6.1.3 Presentation Tier

The presentation tier is the topmost tier of the application and provides the interface by which users engage with the system. There are many different options for the architecture of this tier. The standard in web development has been to dynamically generate HTML code to be displayed in a browser when a request is sent by a user. This is usually done on the same server that implements the application tier. In this case, the server often follows an internal architecture called model-view-controller (MVC) to separate the concerns of different tiers.

An alternate approach is to separate the presentation logic from the application server. To do so, the application server does not return web pages but instead resources in a standard exchange format such as JSON or XML. If the interface adheres to additional constraints, it is called a representational state transfer (REST) API. The presentation logic is then part of a different component, often a JavaScript application running in a user's browser.

It is common for such an application running the browser to request resources from the application tier independent of the page that is currently being displayed. The page is then dynamically generated in the browser and displayed to the user. This type of application is called a single-page application (SPA).

Advantages of this approach include that different application and presentation components can make use of the same interface. A program automatically importing sitelog documents into the system could use the same API that the presentation tier uses. In addition, because page reloads are avoided in the browser, there is improved speed and user experience compared to web applications using server-side generated pages.

Disadvantages include that the application must be downloaded in its entirety the first time it is used and users must have the resources to do this. Furthermore this type of application naturally has poor search engine optimization (SEO). SEO is a measure of how highly search engines rank a website and depends on factors such as performance and accessibility. Search engines employ crawlers to determine these metrics and to index pages. Crawlers do not interact with JavaScript, relying on markup only. SPAs typically do not contain any markup, since everything visible to a user is stored in memory at runtime, making it difficult for SPAs to have high SEO. This can be partially alleviated by using XML sitemaps.

In the case of the current system, the presentation tier is a single page application (SPA) implemented in React. React is a popular JavaScript library for building user interfaces. It has a strong community, and the ability to develop SPAs quickly and efficiently is one of its main benefits.

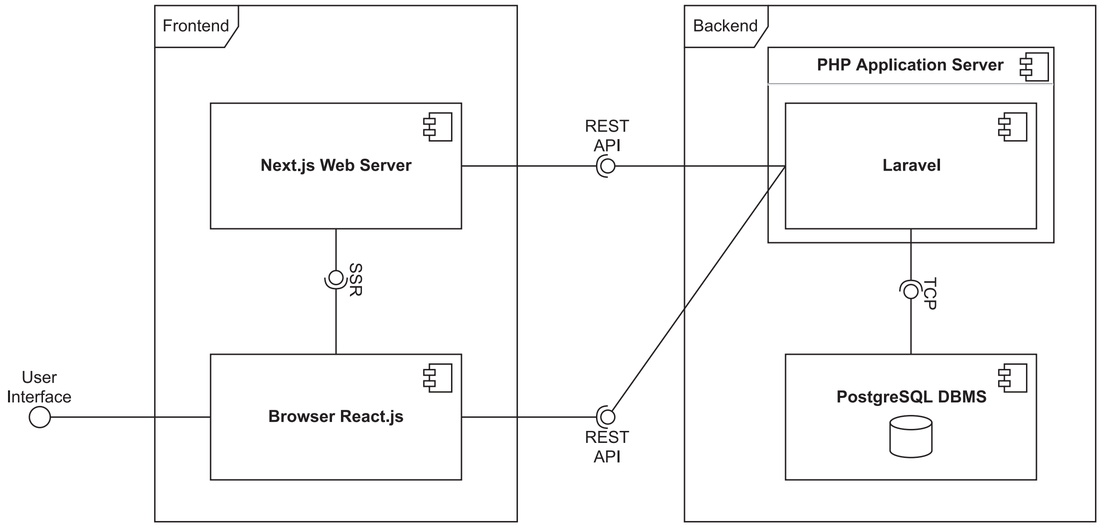


Figure 6.1: Application architecture as component diagram. Data and application tiers are grouped in the "Backend" frame, while the presentation tier is in the "Frontend" frame.

To address the problems of SEO and initial load time, a framework called Next.js is used alongside React to render static markup for React components at build time. When a user or crawler initially enters the application, it is served the generated static markup from the Next.js web server. After the markup is loaded, it is made interactive (*hydrated*) using JavaScript code. The application then transitions to being single-page and browser only. Besides enabling crawlers to index the site, this also yields better largest contentful paint (LCP) time for users compared to pure SPAs, a metric that measures the loading performance of a site.

6.2 Database Schema

A relational database system is a software system that enables users to define, create, control and maintain access to a relational database [21]. Fundamental to relational databases is the relational model. It organizes data into one or more tables of columns and rows. Each table row is uniquely identified by a table-fixed tuple of values called a *primary key* (PK) and other tables may contain references to it as a *foreign key* (FK). In addition to values of each row having table-fixed data types such as number, date or piece of text, they may be marked as nullable (N), where null indicates the absence of a value. A specific database's structure is described by a realization of the relational model called a *database schema*.

The main purposes of a database are efficiency in storage and retrieval of data and maintenance of data integrity. This includes maintaining accuracy and consistency of stored data. In addition to integrity constraints such as primary and foreign keys, database schemas are *normalized* to ensure data consistency. The goal of normalization is to reduce duplicated data as much as possible, such that when a change occurs in the data modeled by the system, a mini-

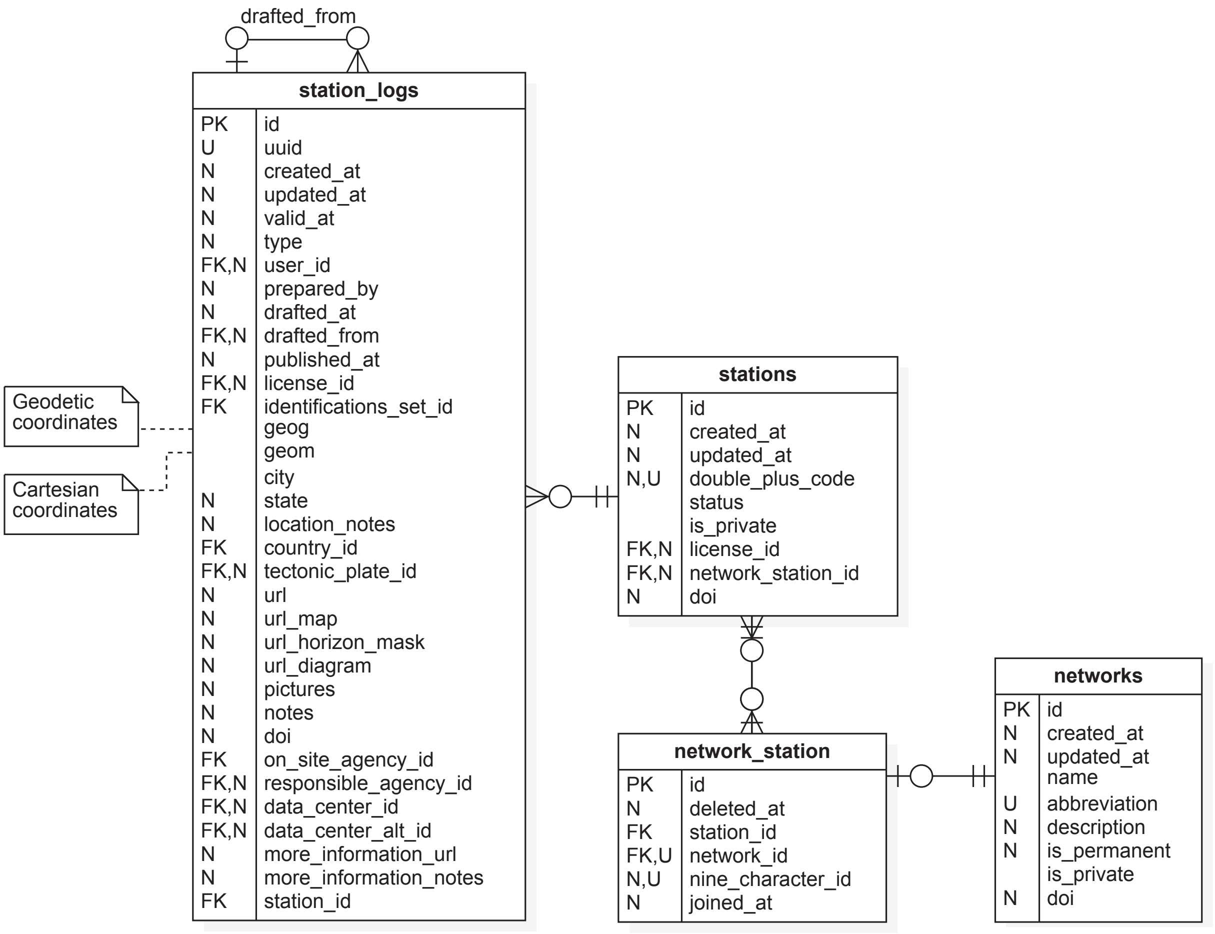


Figure 6.2: ERD of the station_logs, stations, network_station and networks tables.

mum number of changes have to be made within the system to maintain its accuracy. There are varying degrees of normalization, each supplying the relational model with a different set of constraints of varying strictness, as normalization comes at the cost of efficiency. In the current database schema, tables were normalized up to the third normal form.

6.2.1 Station Logs

For a high-level overview of the system's database schema, it is useful to consider the model of a station log, being central to the operation of the system. Station logs are represented by the station_logs table and identified in the system with a universally unique identifier (UUID). They bear a large number of relations to other tables that are used to represent the contents of a sitelog as outlined in 3.1 and 3.2. A superset of the information contained in an ASCII sitelog and a subset of that additionally possible in a GeodesyML sitelog is stored in the current schema.

A station log is many-to-one related to stations, tectonic plates, users, licenses, identification sets, countries, organisations (via data center and alternate data center) and agencies (via on-site agency and responsible agency). It is many-to-many related to agencies via third party agencies. Inversely, it is one-to-many related to surveyed local ties, station sensors, station antennas, station receivers, collocations, local ongoing conditions, frequency standards, other instrumentations and local episodic effects.

Each station log is associated with a single station, for which it represents a snapshot in time of that station's metadata. This is in contrast to previous systems that did not separate the notions of a station log and a station, directly associating a station with its metadata and making it difficult to track changes of a station's metadata over time. Separating them allows adding a drafting and publishing mechanism for station logs with relative ease. A station log is considered to be a draft if its published_at field is null. A station has many logs, some of which can be drafts.

6.2.2 Stations and Networks

Every station belongs to one or more networks via the network_station pivot table. A standard way to identify stations as used in the IGS has been the Nine-Character ID of a station, seeing widespread usage. The Nine-Character ID of the main station at GFZ Potsdam for example is POTS00DEU. This is composed of a four-character string naming a station ("POTS"), a monument number ("0"), a receiver number ("0") and an ISO 3166-1 alpha-3 code, identifying the country a station is located in. However, this identifier is only unique within the IGS network and there are examples of different stations having the same Nine-Character ID outside of it. To address this issue, the Nine-Character ID of a station is scoped to a particular network by storing it in the network_station pivot table.

One of the networks of every station is its master network. A master network assigns permissions to users authorized with the network on every member station. GFZ contributes 24 of its 55 permanent stations to the IGS. The master network in this case is GFZ, having 55 stations, 24 of which are also part of the IGS network.

6.2.3 Double Plus Codes

Part	Description	Regex
9F4M93H8+	Length-8 plus code defining bounding box with 275 m side length at the equator	[2-9CFGHJMPQRVWX]{8}\+
GNS+	Identifier for the geodetic method	[A-Z]{3}\+
001	Identifier of a station within the bounding box	(?!000)[0-9A-Z]{3}

Table 6.1: Structure of a double plus code by example of 9F4M93H8+GNS+001 (POTS00DEU).

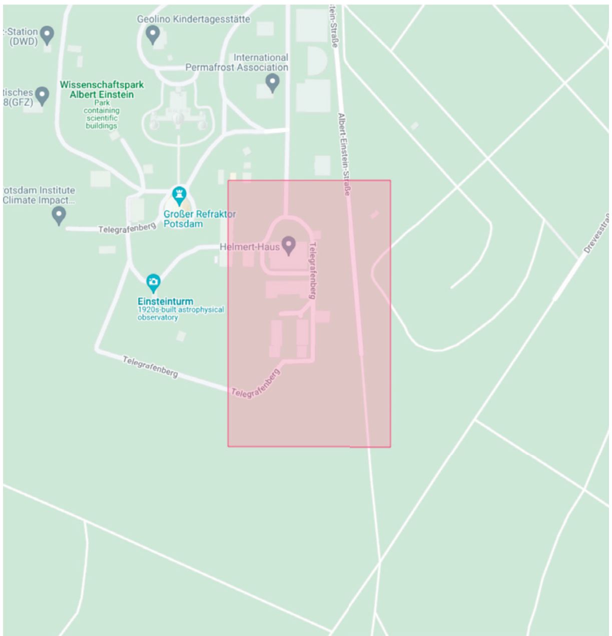

Geolino Kindertagesstätte
-Station (DWD)
International Permafrost Association
ein-Straße
Wissenschaftspark Albert Einstein
Park containing scientific buildings
tisches 8(GFZ)
Albert-Einstein-Straße
otsdam Institute Climate Impact...
Großer Refraktor Potsdam
Telegrafenberg
Helmert-Haus
Telegrafenberg
Drevesstra
Einsteinturm
1920s-built astrophysical observatory
Telegrafenberg
Telegrafenberg

Figure 6.3: Bounding box of 9F4M93H8+GNS+001[1].

The double plus code of a station is a novel application of Google's Open Location Code to the domain of GNSS stations developed within this project. It may be used as an alternate identifier to a station's id field. Open Location Code is a geocode system for identifying an area anywhere on Earth [2] and was specifically designed for use in areas where there is no street addressing. Codes created using this system are called "plus codes". Since currently no identification scheme exists for stations of geodetic and other scientific methods that is unique across networks, easy to obtain and having general consensus, a method using the Open Location Code can be applied in pursuit of these goals.

Plus codes are a variable-length sequence of characters from a set of 20 which consists of selected letters and digits. Characters with potential to cause ambiguity are purposely omitted from the set. Alternating elements encode latitudes and longitudes respectively in WGS 84 coordinates, where each additional pair divides the bounding box defined by the previous into a 5 by 4 grid. A plus code therefore acts as a spatial partitioning tree. If the code is longer than 8 elements, a + character is added after a length of 8 to distinguish the code from regular postal addresses.

Double plus codes consist of a length-8 plus code, an identifier for the geodetic or scientific method and a length-3 identifier within the bounding box defined by the plus code, separated by + symbols (see 6.1 for an explanation by example). This way, every geodetic or scientific method can assign up to 46.655 unique stations within the specified bounding box. Considered geodetic methods are GPN (GNSS), GLR (SLR), GDO (DORIS) and GVL (VLBI). The current

[2] https://plus.codes/

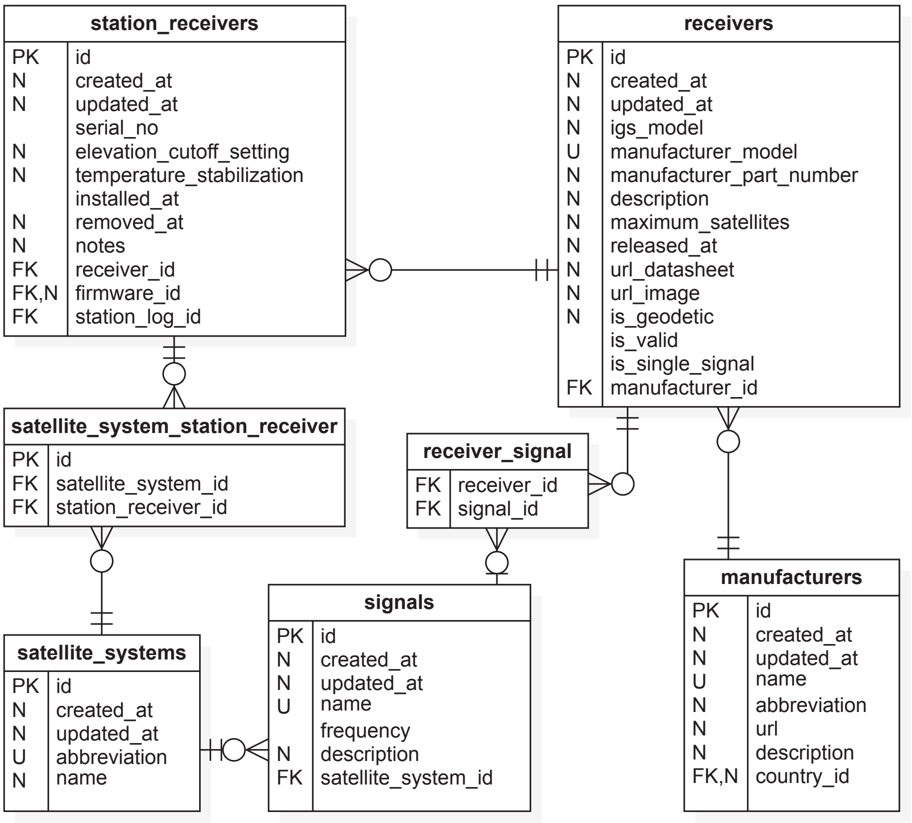


Figure 6.4: ERD of station_receivers and related tables.

system is meant to serve as a reference implementation for double plus codes and to help their adoption by providing a simple to use service for assignment of these codes.

Advantages of double plus codes include easily being able to determine spatial proximity of stations. Since the code is hierarchically structured based on location, stations that are physically close will be close in an alphabetically sorted list. This is in contrast to other identification methods such as Nine-Character ID or IERS Domes Number which reference administrative regions.

6.2.4 Hardware

Station hardware is stored in association with station logs. A station log has many receivers, antennas, frequency standards and meteorological sensors, represented by the station_receivers, station_antennas, station_met_sensors and frequency_standards tables. A receiver installed at a station (station receiver) references a receiver model (receiver). A receiver model is then associated with a manufacturer. Installed antennas and receivers are stored with the same table structure. Every model that a station log is one-to-many related to is stored in a similar way.

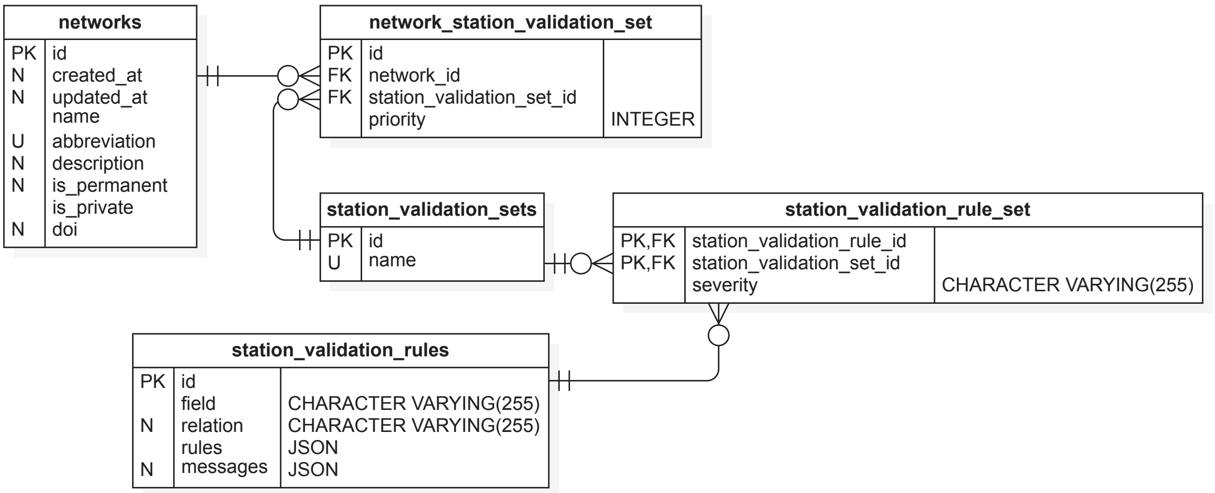

networks
PK id
N created_at
N updated_at
name
U abbreviation
N description
N is_permanent
is_private
N doi
network_station_validation_set
PK id
FK network_id
FK station_validation_set_id
priority INTEGER
station_validation_sets
PK id
U name
station_validation_rule_set
PK,FK station_validation_rule_id
PK,FK station_validation_set_id
severity CHARACTER VARYING(255)
station_validation_rules
PK id
field CHARACTER VARYING(255)
N relation CHARACTER VARYING(255)
rules JSON
N messages JSON

Figure 6.5: ERD of tables related to validation of station logs.

6.2.5 Validation

Validation of station logs is an important part of the system. The networks that a station belongs to provide validation rules for the logs of the station, stored in the station_validation_rules table. A validation rule consists of a field, a relation, rules and messages. Field is the name of an attribute of a model to be validated. Relation is a string identifying how the model to be validated is related to a station log, e.g. ”stationReceivers.receiver” if receiver models are being validated. Rules is a JSON-encoded array of Laravel validation rules[3] to be applied to the field. Finally, messages is a JSON-encoded associative array of rule names to human-readable strings describing the validation error, which can be used to explain validation errors on the frontend.

Validation rules are aggregated by validation sets via a pivot table containing a severity for each rule. Currently, values ”error” and ”warning” are used. Based on a rule’s severity, its error message can be shown differently on the frontend. Validation sets are in turn attached to networks via a pivot table containing a priority for each set. Priorities are used to choose which rule to apply when multiple validation sets attached to a network contain rules on the same field. In this case, it is the rule whose validation set has a higher priority on the network.

[3] https://laravel.com/docs/9.x/validation#available-validation-rules

Chapter 7

Implementation

7.1 Methodology

For the development a framework similar to Scrum was adopted. Scrum is a widely used project management framework for small teams where project goals are broken down into time-boxed iterations called *sprints*. These are typically two weeks in length, every day of which a short stand-up meeting is held. At the end of each sprint lies a review to demonstrate the work done and a retrospective to reflect on the process. Each team consists of a Scrum master and several developers, where the Scrum master mediates between the product owner and the developers.

In the current project, the team size was limited to one person. For this reason, no daily stand-up meetings were held, but regular communication was provided via the project management platform Asana and the GFZ-internal instance of the source code host GitLab. Review meetings were held on a weekly basis to discuss changes made and receive additional input. This way, it was possible to adapt to changing requirements, such as proposed changes in the GeodesyML format. A Kanban board was used to keep track of bugs and proposed features.

7.2 Timeline

Work on the current version of the project was started in late February 2022, with an expected completion date of the minimum viable product (MVP) in August 2022.

7.3 Backend

Work was initially started on the components in the backend part of the application, consisting of database and application server components. As per the technical requirements, the PHP-based web framework Laravel [1] is used to implement the application server.

To set up the development environment, a component of the Laravel software ecosystem called Laravel Sail was used. It is a frontend to the popular containerization service Docker. Docker is a daemon that builds and runs *containers* in which software runs in isolation from the rest of the system. Containers are similar to chroot jails but also have cgroups and networks separate from the host. Containers are built upon *images* which define the initial state of a container's file system. Laravel Sail automatically creates these containers as required by an application. Advantages of a containerized approach include reproducibility of the development environment, which enables other developers to quickly set up a local instance of the software. Containers are not typically used in production, where software is ran on bare metal instead. For the current system, containers were created for PostgreSQL and for Laravel itself running on an Apache HTTP server.

Laravel is a highly object-oriented framework and conveniently provides an object-relational mapper (ORM) called Eloquent for interacting with different DBMSs. An ORM is responsible for representing tables in a database as classes and subsequently rows of those tables as objects in an object-oriented environment. By utilizing the ORM, little to no SQL code as the usual interface to a DBMS had to be written during development, positively impacting developer productivity.

Laravel provides additional facilities closely connected to Eloquent for the creation of tables. The definition of a table schema is called a ″migration″ in Laravel, whereas a class filling a table with values is called a ″seeder″. Both can be used together with the ″artisan″ command to automatically drop all tables in a database, create new tables and fill them with values, making database state easily reproducible. This also requires no SQL code. These facilities were used to create migrations and seeders for all the tables required by the application. Classes were created for use with the ORM corresponding to select tables.

7.3.1 REST API

An API is considered to be *RESTful* if it adheres to the interface standards defined by REST. In practice, this means that correct HTTP verbs are used to interact with server state and server state is expressed in terms of resources. The HTTP specification defines 9 verbs, the most important of which are GET, POST, PUT and DELETE. A resource is anything that has an identifier (URI). To illustrate, an API that requires clients to make POST requests to obtain a resource is not RESTful, as POST requests are defined to not be idempotent, i.e. yield the same result on successive requests. In such a scenario, a client could not be sure that the state of the

[1] https://laravel.com/

server remains unchanged.

Laravel is originally a model-view-controller (MVC) framework. In an MVC architecture, the controller receives input from a client and passes it to the model managing the data of the application. The model then updates the view, which renders a representation of the model and is returned to the client. Since the system's architecture separates the frontend and the backend of the application, only minimal use is made of Laravel's view components. Laravel provides facilities called resource controllers for bulk definition of routing related to resources in the sense of REST. These are classes that implement a subset of the index, store, show, update and destroy methods, which are mapped to the corresponding HTTP operations and assigned URLs under the prefix of the resource. Resource controllers were used to implement the REST API as defined in the requirements and used on the frontend.

The REST API features two authentication schemes. As per the requirements, authentication with Bearer tokens is supported. Requests coming from the frontend are authenticated via session cookies instead. Both authentication schemes are implemented using Laravel Sanctum[2]. Laravel allows the definition of global and route-specific *middleware* stacks. A middleware stack is a list of objects that each perform operations on a request object before it is passed to the next object in the list. This is similar to the chain-of-responsibility design pattern. Sanctum provides middleware for extracting tokens and session cookies and authenticating a user with them before a request reaches the handler method of a target controller.

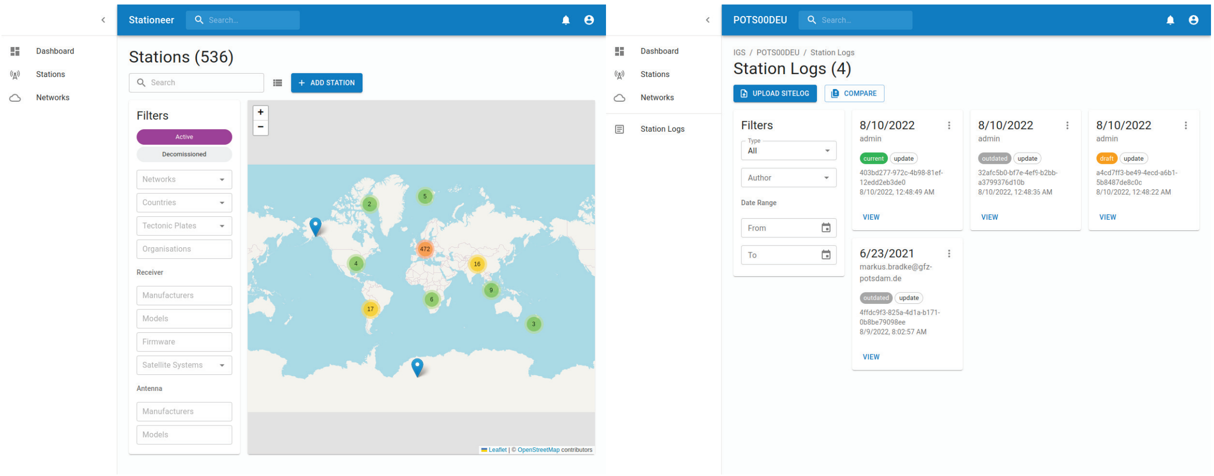
7.3.2 GraphQL API

GraphQL is a schema for the definition of APIs similar to REST. It was developed by Meta (formerly Facebook) and released as open-source in 2015. The main differences between REST and GraphQL are that GraphQL only sends HTTP requests to a single URL endpoint using the POST verb, identifying resources using types and fields of those types instead. It uses a special query language to achieve this. Similar to standard REST APIs, requests and responses are encoded as JSON. Advantages of using GraphQL include type safety in both requests and responses. Disadvantages include the added complexity of having to maintain the schema defining the types used by a GraphQL API.

Early in development, GraphQL was considered for implementing the API but ultimately not used in favor of REST. Within the scope of the project, a PHP package[3] was developed for use with Laravel to generate GraphQL schemas automatically from ORM classes (Laravel Eloquent models). A GraphQL API may still be added in future iterations, especially for better

[2] https://github.com/laravel/sanctum

[3] https://packagist.org/packages/brausepulver/laravel-eloquent-to-graphql



8/10/2022
admin
current update
403bd277-972c-4b98-81ef-12edd2eb3de0
8/10/2022, 12:48:49 AM

8/10/2022
admin
outdated update
32afc5b0-bf7e-4ef9-b2bb-a3799376d10b
8/10/2022, 12:48:35 AM

8/10/2022
admin
draft update
a4cd7ff3-be49-4ecd-a6b1-5b8487de8c0c
8/10/2022, 12:48:22 AM

6/23/2021
markus.bradke@gfz-potsdam.de
outdated update
4ffdc9f3-825a-4d1a-b171-0b8be79098ee
8/9/2022, 8:02:57 AM

(a) Filter and map of stations.

(b) View of the logs of a station.

Figure 7.1: Examples of pages on the frontend.

managing of complex queries.

7.3.3 Permissions

Permissions are implemented using roles and abilities on those roles. A user obtains permissions by being assigned roles. A PHP package called Bouncer ^{4} was used to bootstrap the implementation. Bouncer does not natively support restriction of roles to specific models, such as restricting a ”network administrator” role to a single network, which is required, since every network’s roles must be isolated from those of other networks. This was implemented by using the ”scopes” feature provided by Bouncer and setting the scope of all roles restricted to a model to that model’s name and its id. Passing a ”scope” to Bouncer then restricts any authorization tests to consider only those roles and abilities within the given scope.

7.3.4 GeodesyML

Since GeodesyML is treated as an interchange format, the internal representation of station metadata in the current system is only partially modelled after the schema. Conversion from the internal representation to GeodesyML happens using the SimpleXMLElement class of the PHP standard library. Using SimpleXMLElement, a GeodesyML document is built up incrementally by adding children to a DOM tree. It can then be validated and exported as an XML string.

7.4 Frontend

The frontend was implemented in React[5]. React is a popular library for building users interfaces both in the browser and on mobile devices. React is maintained by Meta and was initially released in 2013. React is most often used in SPAs, which is also the case for the current system. In an SPA, only a single page is requested from a server during a session, although often SPAs provide a similar user experience to traditional web applications with many pages.

7.4.1 Next.js

The foundation of the frontend application is a framework called Next.js[6], which adds many additional features on top of React that make development more similar to a traditional web-application, such as file-system routing, static generation and server-side rendering.

React applications typically go through a process called *building* before being ran in production mode. Internally this *minifies* the code, making it as concise as possible, and *transpiles* it to a version of JavaScript supported by all target browsers. At this stage, Next.js also generates static markup for every page. When a user loads a page in the browser that is part of a Next.js application, she is first served this static markup which is then hydrated with JavaScript.

If a page requires external resources to show much of any content at all, this is not particularly useful. Next.js provides two methods of hooking into the process of generating markup for a page. In both, a page defines its data requirements in a separate function, which is then executed by Next.js when markup is being generated. This way, dynamic data can be included in the markup generated for a page.

The first of these methods is static generation (SG). SG happens at build-time. In addition to defining its data dependencies, a page also defines the space of possible values for dynamic components in its path. SG is used for pages whose contents rarely change, as markup is only regenerated when the application is built again. To make SG more dynamic, it can be used together with incremental static regeneration (ISG), in which the Next.js server provides an endpoint that triggers a regeneration of the static markup of a page when a request is received.

Server-side rendering (SSR) is the second of these methods. It is similar to static generation but happens on every request. SSR is only used when pages need to be pre-rendered for a specific purpose and the dynamic data required cannot be known in advance, as otherwise client-side fetching of dynamic data may be used instead.

The current system uses static generation for public-facing pages that rarely change, such

[4] https://github.com/JosephSilber/bouncer

[5] https://reactjs.org/

[6] https://nextjs.org/

as the landing page, news articles and informational pages. For all other pages, client-side data fetching is used instead.

7.4.2 TypeScript

The frontend was implemented entirely in TypeScript, a strictly syntactical superset of JavaScript that enhances it with static types. It was developed by Microsoft and first released in 2012. Static type checking aids in finding bugs and is especially relevant for large applications like the current system. TypeScript is dissimilar to other statically typed languages such as C++ or Java in having complete type erasure at runtime. While other languages retain some notion of the type of a value, TypeScript is transpiled to JavaScript at build-time, which has no concept of types beyond primitive values and objects.

7.4.3 User Interface

The user interface was built using a component library called MUI[7] that implements the Material Design 2 guidelines. Material Design is a design language developed by Google in 2014. It is based on scientific research in user experience design and provides a wide variety of components for applications to use. It is prominently featured on Android phones. Using Material Design positively impacted development speed as less time had to be spent on manual styling. Still, components were customized to fit the needs of the application and combined to create new components. In the future, the application will receive a visual overhaul using MUI's theming features.

7.4.4 Major Packages

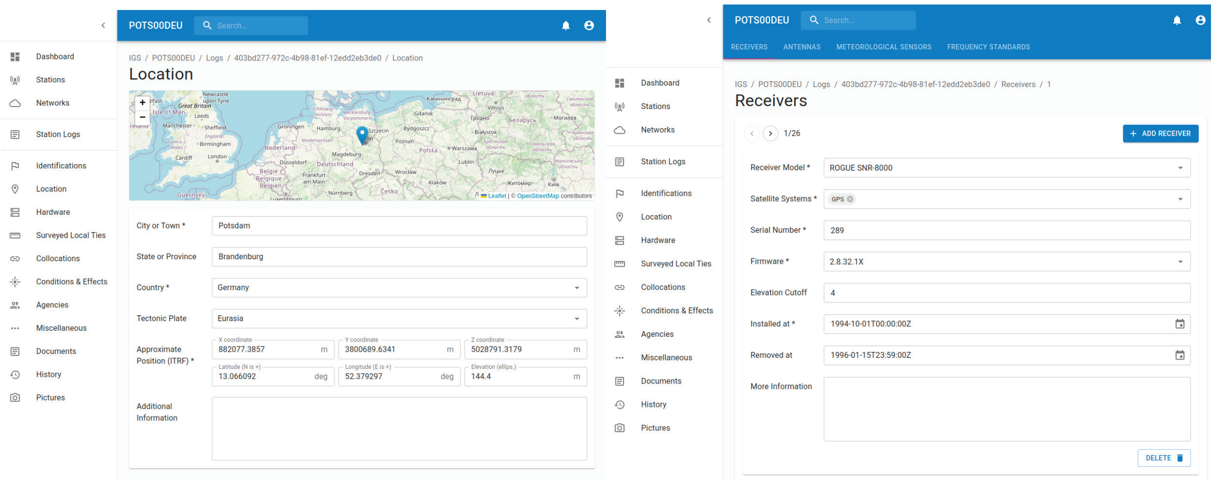
The majority of the pages in the application contain forms of some kind. The package Formik[8] was used to handle form state. The package yup[9] was used to provide validation schemas to Formik such that invalid form entries can be determined. These validation schemas were also partly used to check API responses for correctness. Maps were created using the popular Leaflet[10] library and OpenStreetMap data. In later iterations, a GFZ-internal Web Map Service (WMS) may be used instead.

[7] https://mui.com/material-ui/

[8] https://formik.org/

[9] https://github.com/jquense/yup

[10] https://leafletjs.com/



IGS / POTS00DEU / Logs / 403bd277-972c-4b98-81ef-12edd2eb3de0 / Location

Location

City or Town * Potsdam

State or Province Brandenburg

Country * Germany

Tectonic Plate Eurasia

Approximate Position (ITRF) *

X coordinate	Y coordinate	Z coordinate
882077.3857 m	3800689.6341 m	5028791.3179 m
Latitude (N is +)	Longitude (E is +)	Elevation (ellips.)
13.066092 deg	52.379297 deg	144.4 m

Additional Information

(a) Editing location information of a station.

IGS / POTS00DEU / Logs / 403bd277-972c-4b98-81ef-12edd2eb3de0 / Receivers / 1

Receivers

Receiver Model * ROGUE SNR-8000

Satellite Systems * GPS

Serial Number * 289

Firmware * 2.8.32.1X

Elevation Cutoff 4

Installed at * 1994-10-01T00:00:00Z

Removed at 1996-01-15T23:59:00Z

More Information

(b) Editing receivers of a station.

Figure 7.2: Examples of pages containing forms.

Chapter 8

Results

The majority of the functional requirements laid out in chapter 4 were able to be implemented to date (see table 8.1 for an analysis). Requirements not yet implemented that are part of the MVP will see further development. Important features will be thoroughly tested before the application reaches production stage. On the backend, endpoints related to roles and permissions will be integration-tested using PHPUnit[1]. On the frontend, important use cases will be end-to-end-tested using Cypress[2]. End-to-end tests are tests that involve the entire stack of an application, including frontend, server and backend.

Non-functional requirements can be tested using Lighthouse[3]. Lighthouse is a tool developed by Google that audits a web page for performance, accessibility, best practices, SEO and other criteria generating a score on a scale of 0 to 100 for each.

Performance Lighthouse measures the Core Web Vitals [22] and produces a weighted sum. The Core Web Vitals include metrics such as Largest Contentful Paint, First Input Delay and Cumulative Layout Shift, which have direct influence on how performant a site feels to a user.

Accessibility Pages are scored based on their user experience for visually impaired and screen reader users.

Best Practices Problems such as security issues or JavaScript errors decrease this score.

SEO Pages are scored based on how well crawlers can navigate them and locate content.

These metrics can also be used to compare the system to existing systems as described in chapter 5. Performance scores were obtained by averaging three measurements in each

[1] https://phpunit.de/

[2] https://www.cypress.io/

[3] https://developer.chrome.com/docs/lighthouse/overview/

Table 8.1: Status of the implementations of functional requirements.

Requirement	Status	Description
GUI and API Access	●	
Responsive	◐	Generally complete but some components need to be altered for mobile usability, e.g. filters moved from a page into a drawer.
Authentication	●	
Authorization	◐	Implemented for networks but not stations.
Storage	●	The database schema is updated to meet changing requirements but generally complete.
Validation	◐	Validation is not yet configurable in the GUI.
Updates	○	
API	◐	Generally complete but lacking some minor endpoints. The OpenAPI documentation is not yet complete.
GUI	◐	The majority of use cases can be achieved through the GUI.
SEO	◐	Dependent on completion of the GUI.
Inputs	◐	ANTEX, ocean loading data and station visit logs not yet possible to be imported.

● Fully implemented
◐ Partially implemented
○ Not implemented

case to account for random errors in network latency. The performance measurements are biased by the network latency caused by the distance from the geographic locations at which measurements are taken to the web server. Accessibility, best practices and SEO metrics are unlikely to change between page reloads and therefore not averaged. The measurements were conducted using Lighthouse 9.5.0 using default settings.

The source code for the system will be made available at https://git.gfz-potsdam.de/gnss/gnss-station-meta.

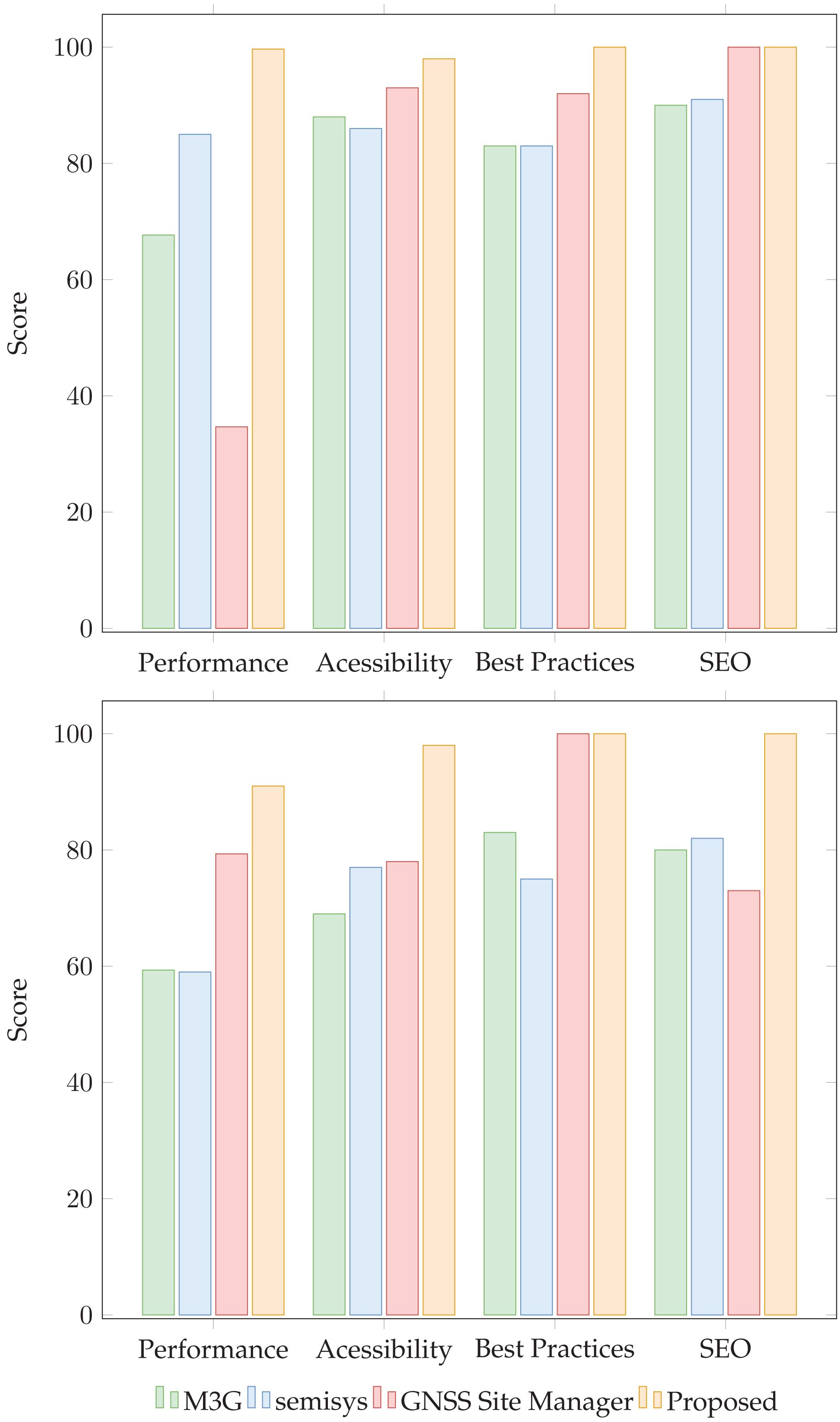


Figure 8.1: Lighthouse metrics for different systems as measured on two pages: a page for viewing and/or editing the log of a station and a page with a map of stations.

Chapter 9

Conclusion

The system proposed here is capable of meeting the needs of a changing GNSS user segment. Full support for the GeodesyML standard enables GNSS station metadata stored in the system to adhere to the FAIR principles and facilitates machine-to-machine exchange that ensures metadata is current and correct. We showed that designing a system with GeodesyML in mind eases implementation of the standard. A REST API is provided for interoperability with systems that do not yet support GeodesyML. We also showed that an updated technology stack yields improved performance and usability of the system.

Since development on the system began, work has picked up on the GeodesyML standard within the IGS Infrastructure Committee after being mostly dormant for 6 years. There are new proposals in draft status that the system will have to accommodate. With another system to support GeodesyML, SLM 2.0, being in development at the same time, the format has potential to reach general adoption and be endorsed as a standard by the IGS, phasing out ASCII sitelogs. Any challenges or inconsistencies related to the format encountered during development are upstreamed to maintainers. This way, future development of the system will help make the format more adoptable by other parties.

In the upcoming months, the system will see initial use by a wider audience. As part of a project funded by the European Research Council[1], 80 new GNSS stations will be installed in Greece for monitoring of tectonic plate motions. The metadata for these stations is designated to be self-managed by project members within the system. This will help fix potential issues and lay the groundwork for adoption of the system by other agencies and station operators.

[1] https://erc.easme-web.eu/?p=101042674

Bibliography

[1] European Union, “EUSPA EO and GNSS Market Report,” en-US, p. 216, 2022. DOI: 10.2878/94903. [Online]. Available: https://www.euspa.europa.eu/sites/default/files/uploads/euspa_market_report_2022.pdf (visited on 08/10/2022).

[2] O. Montenbruck and P. Teunissen, Eds., *Springer Handbook of Global Navigation Satellite Systems*, 1st ed. 2017, ser. Springer Handbooks. Cham: Springer International Publishing : Imprint: Springer, 2017, ISBN: 978-3-319-42928-1. DOI: 10.1007/978-3-319-42928-1.

[3] M. Bradke, *Updates from the GeodesyML Working Group*, Jun. 2022. [Online]. Available: https://files.igs.org/pub/resource/pubs/workshop/2022/IGSWS2022_S10_04_Bradke.pdf (visited on 08/07/2022).

[4] IGS Central Bureau, *Instructions for filling out IGS site logs*, Feb. 2022. [Online]. Available: https://files.igs.org/pub/station/general/sitelog_instr.txt (visited on 08/03/2022).

[5] ——, *XXXX Site Information Form (site log)*, Feb. 2022. [Online]. Available: https://files.igs.org/pub/station/general/blank.log (visited on 08/03/2022).

[6] I. Romero, *RINEX - The Receiver Independent Exchange Format - Version 4.00*, en-US, Dec. 2021. [Online]. Available: https://files.igs.org/pub/data/format/rinex_4.00.pdf (visited on 08/09/2022).

[7] IGS Central Bureau, *IGS Site Log Manager*. [Online]. Available: https://slm.igs.org/login.php (visited on 08/03/2022).

[8] A. Fabian, C. Bruyninx, A. Miglio, *et al.*, “M3G - Metadata Management and Distribution System for Multiple GNSS Networks,” 2021, Publisher: Royal Observatory of Belgium Version Number: 4.2. DOI: 10.24414/ROB-GNSS-M3G. [Online]. Available: https://gnss-metadata.eu/landing/m3g (visited on 08/03/2022).

[9] M. Bradke, “SEMISYS - Sensor Meta Information System,” 2020, Publisher: GFZ Data Services Version Number: 4.1. DOI: 10.5880/GFZ.1.1.2020.005. [Online]. Available: https://dataservices.gfz-potsdam.de/panmetaworks/showshort.php?id=9212b781-017b-11eb-9603-497c92695674 (visited on 08/03/2022).

[10] Geoscience Australia, *GNSS Site Manager*. [Online]. Available: https://gnss-site-manager.geodesy.ga.gov.au/ (visited on 08/03/2022).

[11] IGS Central Bureau, "IGS 2021+ Strategic Plan," en, p. 15, 2021. [Online]. Available: https://files.igs.org/pub/resource/pubs/IGS_Strategic_Plan_2021_Final.pdf.

[12] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," en, *Scientific Data*, vol. 3, no. 1, p. 160 018, Mar. 2016, Number: 1 Publisher: Nature Publishing Group, ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. [Online]. Available: https://www.nature.com/articles/sdata201618 (visited on 08/03/2022).

[13] A. Miglio, A. Fabian, C. Bruyninx, *et al.*, "Proposed metadata standards for FAIR access to GNSS data," en, Copernicus Meetings, Tech. Rep. EGU22-11968, Mar. 2022, Conference Name: EGU22. DOI: 10.5194/egusphere-egu22-11968. [Online]. Available: https://meetingorganizer.copernicus.org/EGU22/EGU22-11968.html (visited on 08/07/2022).

[14] C. Bruyninx, A. Fabian, J. Legrand, *et al.*, "GNSS Station Metadata Revisited in Response to Evolving Needs," en, Copernicus Meetings, Tech. Rep. EGU2020-18634, Mar. 2020, Conference Name: EGU2020. DOI: 10.5194/egusphere-egu2020-18634. [Online]. Available: https://meetingorganizer.copernicus.org/EGU2020/EGU2020-18634.html (visited on 08/03/2022).

[15] N. Brown, R. Fraser, and G. Johnston, "Maximising interoperability and discoverability of geodetic products and services," en, p. 16, 2016. [Online]. Available: http://geodesyml.org/wp-content/uploads/2016/03/BROWN-IGS-Workshop-2016.pdf.

[16] W. Kresse and D. Danko, Eds., *Springer Handbook of Geographic Information*, en, ser. Springer Handbooks. Cham: Springer International Publishing, 2012, ISBN: 978-3-030-53124-9 978-3-030-53125-6. DOI: 10.1007/978-3-030-53125-6. [Online]. Available: https://link.springer.com/10.1007/978-3-030-53125-6 (visited on 08/10/2022).

[17] N. Brown, *International Standards and GeodesyML*, en-US, May 2016. [Online]. Available: http://geodesyml.org/international-standards-and-geodesyml/ (visited on 08/07/2022).

[18] C. DeMets, R. G. Gordon, and D. F. Argus, "Geologically current plate motions," en, *Geophysical Journal International*, vol. 181, no. 1, pp. 1–80, Apr. 2010, ISSN: 0956540X, 1365246X. DOI: 10.1111/j.1365-246X.2009.04491.x. [Online]. Available: https://academic.oup.com/gji/article-lookup/doi/10.1111/j.1365-246X.2009.04491.x (visited on 07/31/2022).

[19] M. Bradke, “Konzeption und Entwicklung eines datenbankbasierten Systems zur Verwaltung von GNSS-Daten,” de, M.S. thesis, Hochschule Neubrandenburg, 2012. [Online]. Available: https://digibib.hs-nb.de/resolve/id/dbhsnb_thesis_0000000846?_search=d1e47a24-a251-4fd1-a445-13fadca9cd12&_hit=0 (visited on 08/07/2022).

[20] Statista, *Most popular database management systems 2022*, en, Jan. 2022. [Online]. Available: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/ (visited on 08/01/2022).

[21] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*, eng, 6. ed., global ed, ser. Always learning. Boston Munich: Pearson, 2015, ISBN: 978-1-292-06118-4.

[22] Google, *Web Vitals*, en, 2022. [Online]. Available: https://web.dev/vitals/ (visited on 08/10/2022).

List of Figures

List of Tables

Appendix

9.1 Business Processes

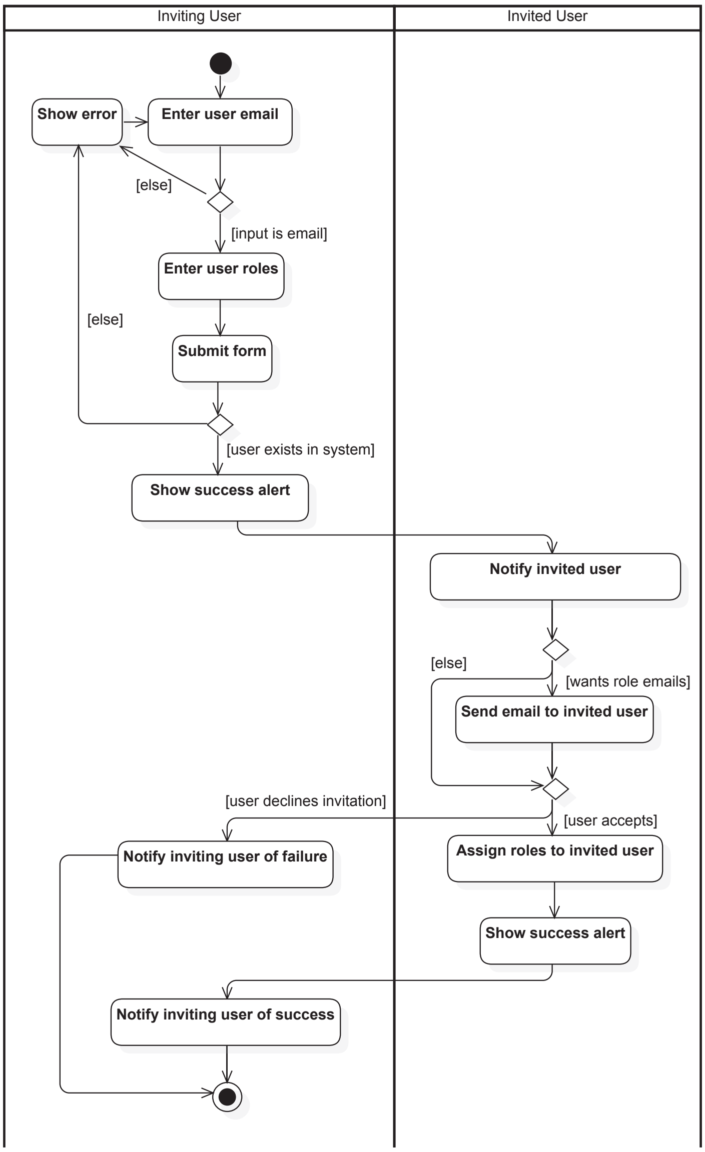


Figure 9.1: Activity diagram for the process of inviting a user to take up a role on a network or station.

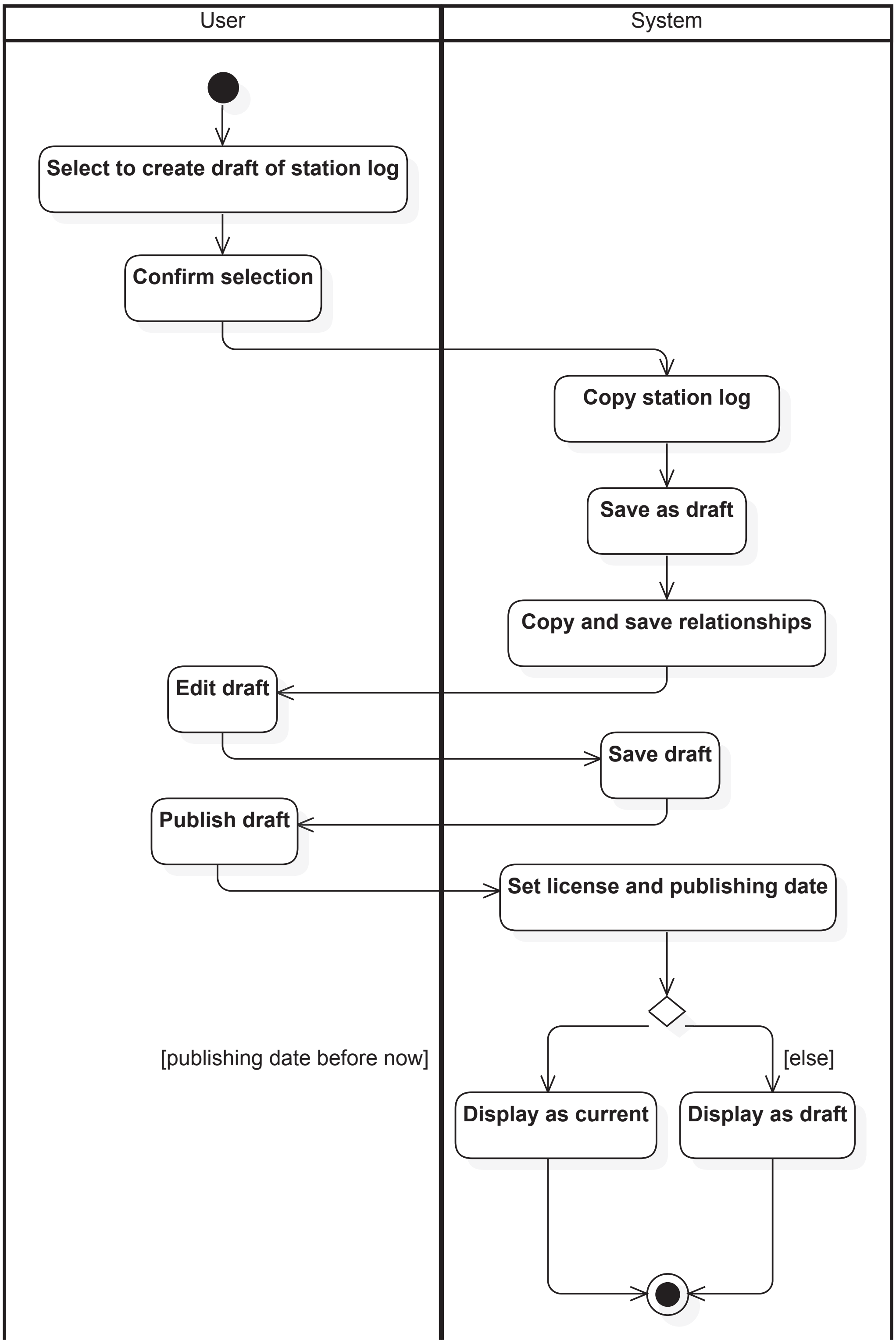


Figure 9.2: Activity diagram for the process of drafting, editing and publishing a station log.

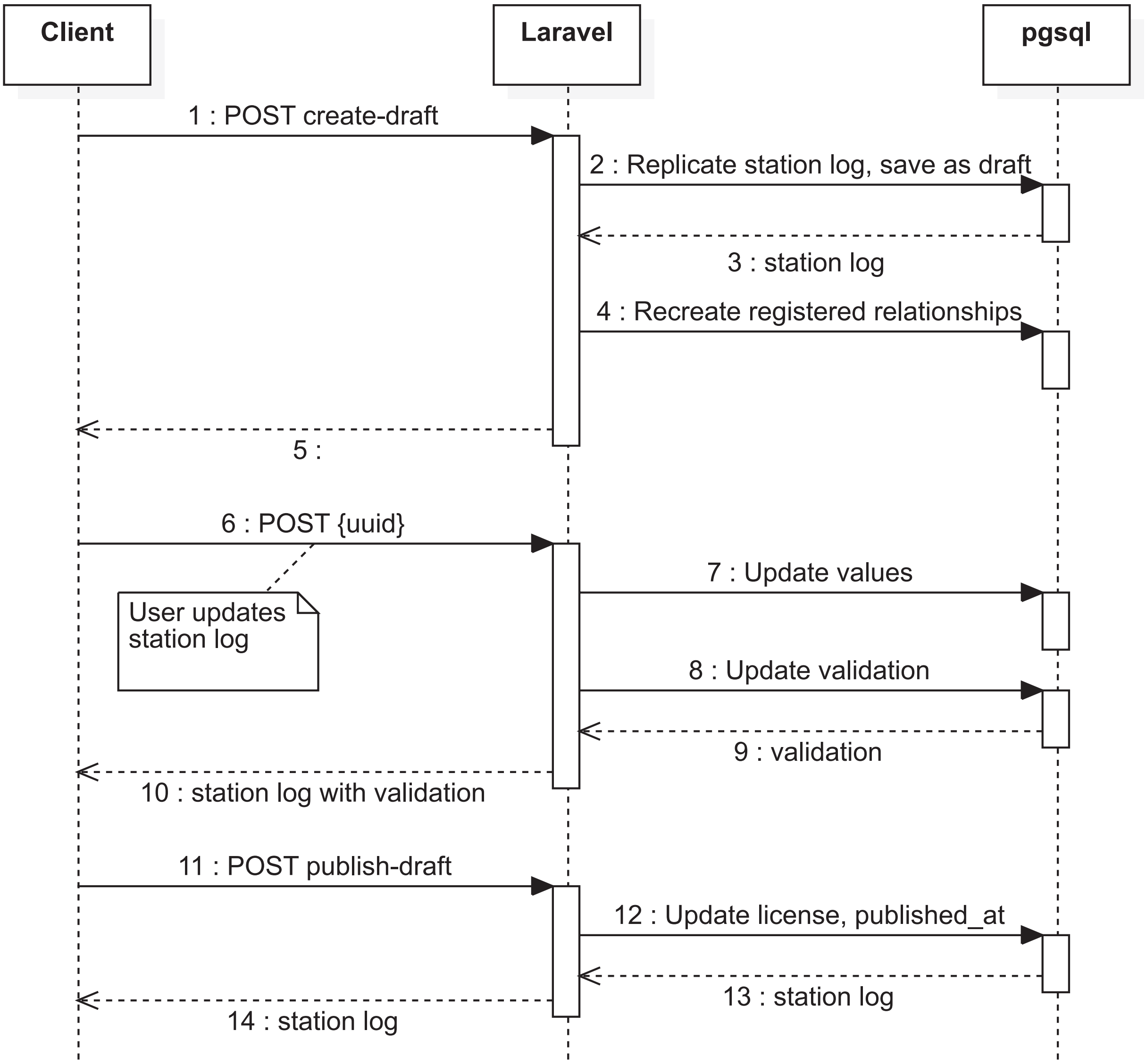


Figure 9.3: Sequence diagram for the process of drafting, editing and publishing a station log.