Chapter 3
Smittestopp Backend

Cise Midoglu, Benjamin Ragan-Kelley, Sven-Arne Reinemo, Jon Jahren and Pål Halvorsen

Abstract An efficient backend solution is of great importance for any large-scale system, and Smittestopp is no exception. The Smittestopp backend comprises various components for user and device registration, mobile app data ingestion, database and cloud operations, and web interface support. This chapter describes our journey from a vague idea to a deployed system. We provide an overview of the system internals and design iterations and discuss the challenges that we faced during the development process, along with the lessons learned. The Smittestopp backend handled around 1.5 million registered devices and provided various insights and analyses before being discontinued a few months after its launch.

C. Midoglu
Department of Holistic Systems, Simula Metropolitan Center for Digital Engineering,
e-mail: cise@simula.no

B. Ragan-Kelley
Department of Numerical Analysis and Scientific Computing, Simula
e-mail: benjaminrk@simula.no

S.A. Reinemo
Simula Metropolitan Center for Digital Engineering,
e-mail: svenar@simula.no

J. Jahren
Microsoft, Norway,
e-mail: Jon.Jahren@microsoft.com

P. Halvorsen
Department of Holistic Systems, Simula Metropolitan Center for Digital Engineering,
Department of Computer Science, Oslo Metropolitan University,
Department of Informatics, University of Oslo
e-mail: paalh@simula.no

A. Elmokashfi et al. (eds.), *Smittestopp − A Case Study on Digital Contact Tracing*,
Simula SpringerBriefs on Computing 11, https://doi.org/10.1007/978-3-031-05466-2_3

3.1 Introduction

The COVID-19 pandemic struck Europe very hard in the spring of 2020. In Norway, manual tracing was successfully used to identify and quarantine close contacts of confirmed cases to some extent, helping reduce the spread of the disease. However, since manual tracing is slow and not necessarily comprehensive in terms of the list of contacts, as well as hard to scale, implementation of a national digital contact tracing solution was considered.

Collaboration between Norwegian Institute of Public Health (NIPH) and Simula Research Laboratory (Simula) was set up to develop an efficient and scalable digital contact tracking solution. On the frontend, this solution would correspond to a mobile application (from here on, referred to as “mobile app" or “app") that could be installed on end user devices such as smartphones and wearables, run in the background without requiring any user interactions after registration, and continuously collect information about the device’s own location as well as other devices in close proximity, using Global Positioning System (GPS) and Bluetooth (BT). Data from the mobile app would be collected centrally on the backend and analysed with an automated pipeline, allowing for the rapid generation of risk reports for every confirmed case. The complete system was named “Smittestopp".

The Smittestopp backend is one of the most crucial parts of the larger Smittestopp system, presented in Figure 3.1. It enables the ingestion and storage of data from the Smittestopp mobile app [15], which is detailed in Chapter 2, handles database and cloud operations for the automated processing and aggregation of these data by the Smittestopp analytics pipeline, which is detailed in Chapters 4 through 7, and interfaces with web applications, which were themselves outside the scope of the project on the Simula side.

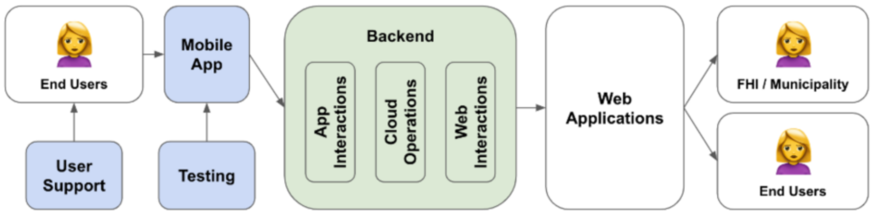


Fig. 3.1: Overview of the Smittestopp system, where coloured components indicate the scope of the technical solution developed by Simula. The focus of this chapter is in green.

The Smittestopp project had two main goals:

- An automatic solution for **contact tracing and notification** (*Varslingsløsning*) with a centralised architecture, for detecting those who have been in contact with

infected individuals. Support for an automated analytics pipeline, detecting and tracing contacts between devices that have the mobile app installed.
- A solution for **aggregated statistics** (*Kunnskapsinnhenting*) that generates information on when and where contacts occured, how many people had encounters, and so on. Functionality was to be developed for tracing the spread of the pandemic throughout the country following national measures, via anonymised aggregations at the population level.

In this chapter, we give an overview of the backend solution put in place to address and support these goals. Overall, the technical requirements from the Smittestopp backend included both functional requirements for meeting the analytics needs and nonfunctional requirements for fast and efficient data management, as well as for protecting personal information. The development of the Smittestopp mobile app and the analytics pipeline were covered in separate work packages and are elaborated upon in different chapters of this book, namely, Chapters 2 and 6, respectively.

The remainder of this chapter is organised as follows: In Section 3.2, we describe the technical implementation of the backend solution, including the architecture, components, and end-to-end operations. In Section 3.3, we discuss how the implementation evolved over the course of five weeks, while being used nationally by up to 1.5 million people, and we describe our experiences and lessons learned. We conclude the chapter in Section 3.4. Readers who are interested in the technical details of our implementation are encouraged to continue reading through Section 3.2, whereas readers who would like to focus on our lessons learned and discussions can skip ahead to Section 3.3. The contents of Sections 3.2 and 3.3 can be studied independently from one another.

3.2 Technical implementation

In this section, we provide a detailed description of the architecture, design, and implementation of the Smittestopp backend, insofar as it was within the scope of the project on the Simula side. We elaborate on the Smittestopp backend in the following manner: In Section 3.2.1, we reiterate the designated high level functionalities required for the overall system. In Section 3.2.2, we provide a technical description of the components and technologies that are employed in the Smittestopp backend to support these functionalities. In Sections 3.2.3 to 3.2.5, we describe the role played by the Smittestopp backend in delivering data from the mobile app to the web applications in a usable manner, in line with the stated goals.

More specifically, in Section 3.2.3, we describe the first stage, denoted as app interactions in Figure 3.1. This comprises a set of functions ranging from device registration and deletion to authorisation, data upload, the handling of incoming data with possible errors, and the transfer of data to the second phase of cloud-based operations. In Section 3.2.4, we describe these cloud-based operations in detail, including data ingestion and storage and running the analytics pipeline. In Section 3.2.5, we elaborate on the higher-level services that the Smittestopp backend provides to connected web applications in the form of the collection of endpoint queries. Further documentation related to the architecture and operations of the Smittestopp backend, please refer to the Smittestopp source code [21].

3.2.1 Required functionalities

3.2.1.1 Contact tracing and notification

One of the main goals of the Smittestopp system was to provide a centralised functionality for notifying people, in case they have been in contact with an infected individual. This is achieved through the use of an automated analytics pipeline detecting and tracing contacts between all devices that have the mobile app installed.

Summarised in Figure 3.2, the functionality works as follows: authorised accounts can use a web application to make queries such as list all close contacts of the user identified by phone number *X*, starting from a given date *T*, covering the past 14 days, where the threshold for a close contact is less than 2m for longer than 15 minutes. The query is serviced by the backend, and a response in the form of a report generated by the analytics pipeline is fed back to the web application.

See Section 3.2.5.1 for more details about the service we implemented to fulfil this functionality, and Chapters 4 through 6 for more details about the detection and reporting of contacts.

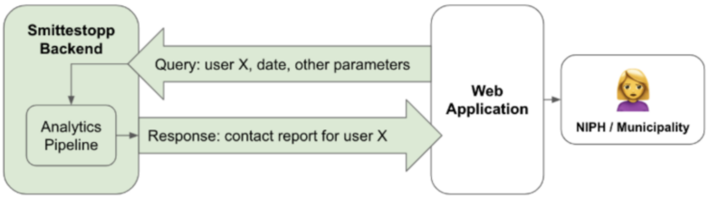

Smittestopp Backend
Query: user X, date, other parameters
Analytics Pipeline
Response: contact report for user X
Web Application
NIPH / Municipality

Fig. 3.2: High-level overview of the contact tracing functionality of the system, where coloured components indicate the scope of the backend solution.

3.2.1.2 Aggregated statistics

The second goal of the Smittestopp system was to provide additional information to support the tracing the of the spread of the pandemic throughout the country, especially following national measures. These are in the form of anonymised aggregations at the population level.

Summarised in Figure 3.3, the functionality works as follows: authorised accounts can use a web application to make queries such as list the number of contacts that have taken place in a specific Point of Interest (POI) (e.g. grocery stores), in a specific municipality (e.g. Oslo), on a given date *T*, with an hourly granularity. The query is serviced by the backend and an anonymised aggregate response, generated by the analytics pipeline, is fed back to the web application. The Points of Interest (POIs) can be any supported item, ranging from healthcare or education facilities to arts, entertainment, and culture or sports or commercial and residential areas [30]. The location of interest can be specified as a county (*fylke*), municipality (*kommune*), district (*bydel*), or basic statistical unit (*grunnkrets*)[1], or custom defined as an area polygon.

It should be noted here that only the NIPH and the government will have the clearance to run an aggregate statistics query. There are also constraints against running aggregate statistics in sparse spatiotemporal contexts, which carry the risk of revealing individual insights.

See Section 3.2.5.2 for more details about the service we implemented to fulfil this functionality, and Chapter 7 for more details about data aggregation and statistics (including privacy preserving techniques and anonymity).

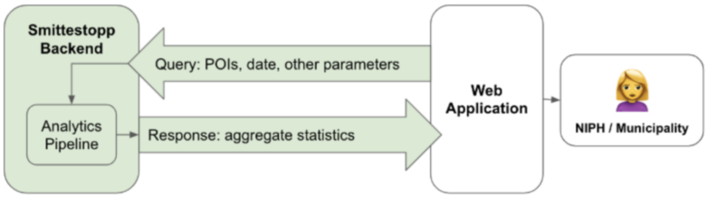


Fig. 3.3: High-level overview of the system's aggregated statistics functionality, where coloured components indicate the scope of the backend solution.

3.2.1.3 User data access

Additional functionality required of Smittestopp for transparency and privacy protection purposes was end user access to their own data. Support for this functionality was in the process of being developed.

Summarised in Figure 3.4, the functionality works as follows: any registered end user account can log in to `https://helsenorge.no` and browse all their GPS-based location data stored in the Smittestopp backend by date and time by using an Application Programming Interface (API) call. Users can also request an access log showing the details of all persons who have viewed their data including the legal means for doing so.

[1] For a full list of area identifiers in Norway, see [1, 26].

It should be noted here that only individual users will have the clearance to run this query, and they will only be able to retrieve their own data. Access to user data in this form and granularity is not available through any other service.

See Section 3.2.5.3 for more details about the service we began implementing to fulfil this functionality.

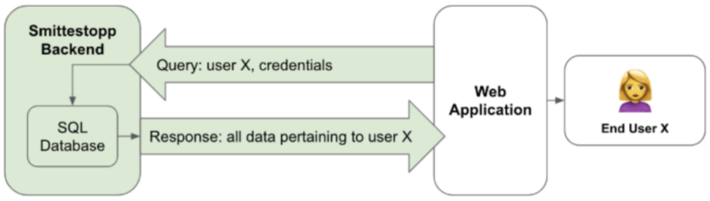

Smittestopp Backend
SQL Database
Query: user X, credentials
Response: all data pertaining to user X
Web Application
End User X

Fig. 3.4: High-level overview of the system's user data access functionality, where coloured components indicate the scope of the backend solution.

3.2.2 Backend components

Figure 3.5 presents an overview of the Smittestopp backend architecture, consisting of three main parts: (1) interactions with the Smittestopp mobile app, (2) cloud operations hosted on Microsoft Azure [10], and (3) interactions with a number of web applications maintained by NIPH and Helsenorge. Below, we provide a technical description of these components.

Note on sandboxing: To support the continuous development and testing of the Smittestopp backend solution, including after the national launch, and in a privacy-preserving manner, all Azure cloud resources (1 – 8) were duplicated into what are called development (`dev`) and production (`prod`) environments. These environments have different access rights, with corresponding mobile app versions targeting different registration services, and two completely different data sets being collected. The data in `dev`, incidentally much smaller in amount than in `prod`, are from informed volunteer testers, do not require anonymised processing, and can be used for research purposes. In addition to NIPH and Norwegian Health Network (NHN) [16] personnel, we recruited volunteers from Simula, along with any of their friends and family members who wanted to support the efforts, through internal campaigns (company-wide announcements) to contribute to the development process by testing different versions of the Smittestopp mobile app on their mobile devices. The valuable efforts from these volunteers complemented the more controlled testing approaches undertaken by the testing team (see Figure 3.1).

Note on performance monitoring: All backend components were instrumented with a monitoring tool called Azure Log Analytics [7]. This service is essentially in place for auditing (who accessed what, when, and why), troubleshooting, security

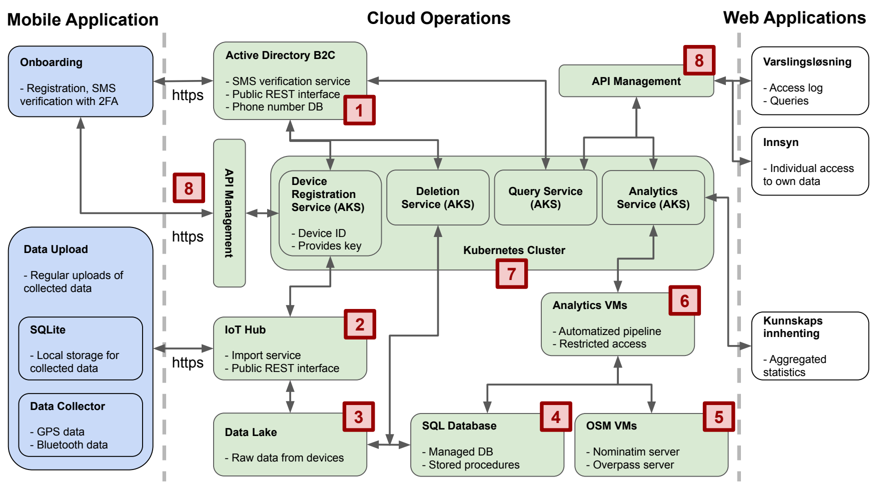


Fig. 3.5: Overview of the Smittestopp backend architecture. Coloured components indicate the scope of the technical solution developed by Simula.

monitoring, and performance monitoring purposes. The log analytics instance collects data from all Azure services, which is stored in a storage container designated by the account holder. Throughout Smittestopp's run, the log analytics instance was accessible only by NHN, and the developers and personnel to whom it explicitly extended access.

3.2.2.1 Active directory B2C

Azure Active Directory B2C (Azure AD B2C) provides Business-to-Consumer (B2C) identities as a service, where the customers can use their preferred social, enterprise, or local account identities to gain single sign-on access to applications and the APIs. It is a Customer Identity Access Management (CIAM) solution capable of supporting millions of users and billions of authentications per day. It takes care of the scaling and safety of the authentication platform, monitoring and automatically handling threats such as Denial of Service (DoS), password spraying, and brute force attacks [13].

All Azure services employed by the Smittestopp backend use the AD B2C enterprise identity service to provide single sign-on and multi-factor authentication. The component is denoted by the number 1 in Figure 3.5. We detail the use of Azure AD B2C in the context of cloud operations in Section 3.2.4.

3.2.2.2 IoT hub

Azure Internet of Things (IoT) Hub is a managed service hosted in the cloud that acts as a central message hub for bidirectional communication between IoT applications and the devices it manages. Virtually any device can be connected to IoT Hub. IoT Hub supports communications both from the device to the cloud and from the cloud to the device, as well as file uploads from devices, and request–response methods to control the devices from the cloud. IoT Hub monitoring allows the tracking of events such as device creations, device failures, and device connections [4].

The Smittestopp backend uses the Azure IoT Hub as a central cloud hosted solution for communication with Smittestopp mobile app clients, and the IoT Hub device registry as the identity provider for each mobile app instance. The component is denoted by the number 2 in Figure 3.5. We detail the use of IoT Hub in the context of the interactions with the Smittestopp mobile app in Section 3.2.3.

3.2.2.3 Data lake

Azure Data Lake Storage is a hyperscale repository for big data analytics workloads and a Hadoop Distributed File System (HDFS) for the cloud. It builds upon all features of Azure Blob Storage and adds a hierarchical namespace for efficient data access across large volumes of files. It imposes no fixed limits on file or account size. Azure Data Lake Storage uses Azure Active Directory (AD) for authentication and protects folders and files using Access Control Lists (ACLs). It allows for unstructured and structured data in their native formats and is tuned for massive throughput on large amounts of files [9].

GPS and BT data uploaded by Smittestopp mobile app clients to the Smittestopp backend passes through the Azure IoT Hub, and from there it is written in batches to a secure blob container in Azure Data Lake Storage Gen2. For troubleshooting purposes, an instance of Azure Search service was also created on top of the Azure Data Lake Storage container, which was designed to facilitate quick searches for event data belonging to a specific device. This component is denoted by the number 3 in Figure 3.5. We detail the use of Azure Data Lake in the context of cloud operations in Section 3.2.4.

3.2.2.4 SQL database

Azure SQL Database is based on SQL server database engine architecture that is adjusted for the cloud environment in order to ensure availability even in cases of infrastructure failure. The hyperscale service tier in is the newest service tier in the virtual core–based (vCore-based) purchasing model. It is a highly scalable storage and compute performance tier that leverages the Azure architecture to scale out storage and compute resources. The Azure SQL Database Hyperscale can scale up to a database of 100 TB, run on up to 80 vCores, and have up to five mirrored readable replicas. With replicas configured, the uptime SLA for the service is guaranteed at 99,99% [8].

The Smittestopp backend uses an Azure SQL Database Hyperscale instance for most of its data analysis operations. This component is denoted by the number 4 in Figure 3.5. Data are ingested into the SQL Database from files in the Azure Data Lake. A major benefit of the hyperscale tier is that it has very short operational delays for reconfigurations and database restores, typically within 10 minutes, regardless of database size. SQL Database Hyperscale also supports index partitioning and compressed columnstore indexes, which provide an estimated compression of 10× on average, with a performance optimisation benefit as well. At peak usage, the Smittestopp database ran on 80 vCores with one readable replica and a database of 4 – 5 TB (compressed).

The Smittestopp database was completely locked down to allow access only to essential Azure services and application servers. All access was monitored by audit logs, and services only had access to the SQL functions they needed, besides two users with administrative access for deployment purposes, controlled by NHN. The component is denoted by the number 4 in Figure 3.5. We further elaborate on the use of the SQL Database in Section 3.2.4.

3.2.2.5 Data factory

Azure Data Factory is the main managed service in Azure for batch ingestion, and has a graphical interface for designing complex data flows and monitoring job executions [14].

The Smittestopp backend orchestrates data movement and performs data wrangling using Azure Data Factory. It uses Data Factory for loading data from Data Lake into Azure SQL Database, in addition to executing SQL stored procedures that perform additional data preparations after each batch ingestion. The data is not actually moved through the Data Factory service, which only executes commands against the underlying services. Azure Data Factory allows past jobs to be re-executed at the task level, which was an advantage for Smittestopp, since the system was not monitored 24/7. Data Factory jobs are run periodically, typically every hour. The service resides between the components denoted by the numbers 3 and 4 in Figure 3.5.

3.2.2.6 Stream analytics

Azure Stream Analytics is a serverless scalable Complex Event Processing (CEP) engine by Microsoft that enables users to develop and run real-time analytics on multiple streams of data from sources such as devices, sensors, web sites, social media, and other applications [12].

Within Smittestopp, the Azure Stream Analytics service was used initially to process incoming events from IoT Hub as data arrived, performing basic validation, filtration and pre-processing and continuously pushing data to the SQL Database.

The service was used during the first week of the launch, after which a decision was made to move to a batch ingestion process via Azure Data Lake Storage instead. The Stream Analytics job was abandoned after the Azure Data Factory jobs were running reliably. The switch took approximately one to two weeks and was accomplished with no downtime. The main reason for the switch was to improve the troubleshooting and end-to-end tracking of individual device events, which proved difficult with the Azure Stream Analytics service. We further detail the use of Stream Analytics in the context of cloud operations in Section 3.2.4 and discuss challenges related to data import, processing, and filtering in Section 3.3.

3.2.2.7 OSM VMs

Map matching involves combining GPS data with metadata from publicly available maps, in order to understand the kind of an environment in which a contact occured (e.g. inside a building, on public transportation, inside a private vehicle, outside), since this could impact the risk level of the contact, as well as the POIs around the contact (e.g. schools, grocery stores, public parks), which could allow for a more informed tracing of the performance of anti-pandemic measures. For the map matching purposes of the Smittestopp analytics pipeline (see Chapter 4 for details), we considered Google Maps [25], Azure Maps [6], and Open Street Maps (OSM) [19], before OSM was selected based on performance, metadata availability, and privacy requirements.

The Smittestopp backend hosts two OSM servers in each environment, (`prod` and `dev`). The OSM servers in the `prod` environment are only reachable from analytics Virtual Machines (VMs) in the `prod` environment. The servers in the `prod` environment use `HTTPS`. The corresponding Transport Layer Security (TLS) certificates contain only the names, and not specific IP addresses. This allows for future changes of the addresses.

- **OSM Nominatim:** This API is a tool to search OSM data by name and address (geocoding) and to generate synthetic addresses of OSM points (reverse geocoding) [28].
- **OSM Overpass:** This is a read-only API that serves up custom selected parts of the OSM map data. It acts as a database over the web: the client sends a query to the API and receives back the data set that corresponds to the query [29].

This component is denoted by the number 5 in Figure 3.5. We further elaborate on the use of the OSM servers in Section 3.2.4.

3.2.2.8 Analytics VMs

The Smittestopp backend hosts two VMs (Linux and Windows) in each environment, to support the development and execution of the Smittestopp analytics pipeline (Chapters 4 through 7). These machines have access to the SQL Database and the OSM VMs and are used to run the pipeline. The analytics VMs in the `prod` environment have restricted access. The component is denoted by the number 6 in Figure 3.5. We further elaborate on the use of the analytics VMs in Section 3.2.4.

3.2.2.9 Kubernetes cluster

Azure Kubernetes Service (AKS) is a cloud hosted service that manages Kubernetes and provides capabilities such as health monitoring, maintenance, and provisioning [5]. The Smittestopp backend uses an Azure Kubernetes cluster for hosting application services, where the following four dockerised services are deployed to AKS:

- **Device Registration Service:** This service handles device registration and consent revocation requests from the mobile app. Also shown are the clients for communicating with the following external services:
  - Microsoft Graph API (access point for information about Azure AD users)
  - IoT Hub (access point for information about devices)
  - SQL Database (GPS and BT data)
- **Deletion service:** This service is responsible for deleting data that is old or associated with users who have revoked their consent.
- **Query service:** This service handles requests from NIPH and Helsenorge. It uses Redis to queue analysis jobs that are processed by the analysis image when requested by NIPH.
- **Analytics service:** This service is responsible for identifying potential contacts between an infected individual and other application users over the time span requested by NIPH. The analysis service communicates with the other services through reads and writes to a Redis database.

The component is denoted by the number 7 in Figure 3.5. Relevant services are detailed in Sections 3.2.3, 3.2.4, and 3.2.5.

3.2.2.10 API endpoints

The Smittestopp backend serves three groups of endpoints, one for each of the following clients:

- Smittestopp mobile app
- NIPH web application
- Helsenorge web application

All Hypertext Transfer Protocol (HTTP) endpoints are managed in Azure API Management (APIM), which is configured via Terraform [3]. The details of the authentication processes vary with the endpoints that are called by the different clients. The mobile app endpoints are behind a Web Application Firewall (WAF), whereas the NIPH and Helsenorge endpoints are accessed directly in APIM (due to a lack of support for SSL client authorisation in WAF). A service prefix is added in APIM, namely, `/fhi` or `/onboarding`. Table 3.1 presents the list of endpoints. The APIM instances are denoted by the number 8 in Figure 3.5.

Type	Endpoint	Description
App	`authentication`	The app authenticates with a JavaScript Object Notation (JSON) web token from Azure AD B2C. Other endpoints authenticate with Hash-based Message Authentication Code (HMAC) signatures using the IoT Hub device ID and signing key.
	`register new device`	Registers a new device with IoT Hub and associates it with an existing user profile.
	`revoke consent`	Revokes permissions granted by a user. All data associated with the user will be deleted.
	`request pin`	Returns Personal Identification Number (PIN) codes for the given user.
	`update birth year`	Updates the registered birth year for the given user.
	`request new bluetooth contact ids`	Allocates and returns 10 new IDs for use as BT contact IDs.
FHI	`lookup phone number`	Requests contact tracing analysis to be performed for the given phone number.
	`lookup result`	Checks results from the contact tracing analysis for a given phone number. Returns results (`200 OK`) if the analysis is ready, or a not finished message (`202 Accepted` otherwise).
	`access log`	Returns the access log for a given user.
	`egress`	Returns the GPS events for a given user in a given time frame.
	`lookup deleted numbers`	Takes a list of phone numbers and returns the numbers not registered in the database.
Helsenorge	`access log`	Returns the access log for a given user.
	`egress`	Returns the GPS events for a given user in a given time frame.
	`revoke consent`	Revokes permissions granted by a user. All data associated with the user will be deleted.

Table 3.1: Smittestopp API endpoints.

3.2.3 Interactions with the mobile app

We now refer to the left part of Figure 3.5, which consists of the interactions between the Smittestopp backend and the Smittestopp mobile app. The components that are involved are numbered as 1 (AD B2C), 2 (IoT Hub), 7 (AKS), and 8 (APIM).

3.2.3.1 Key concepts

The key concepts for the interactions of the backend with the mobile app are as follows:

- A **device** is an instance of the Smittestopp mobile app that has a device ID in Azure IoT Hub and a phone number registered in Azure AD B2C.
- A **user** is defined as a phone number and can be tied to multiple devices.
- **Onboarding** is the process of guiding a new user through the registration process. This includes information about the Smittestopp mobile app, approval of the privacy policy, and entering the required information for registration.
- **Registration** is the process of signing up for Smittestopp using the mobile app. This process includes providing and verifying a phone number using a text message and providing and confirming the year of birth. The registration is described in more detail in Section 3.2.3.2.
- **Authentication** is the process of authenticating a user against one of the endpoints provided by the backend. In the Smittestopp mobile app, authentication is performed when registering a new user, logging in a previously registered user, and when revoking consent.
- **Deletion** is the process of revoking consent for the collection of data and deleting any stored information about the user. In addition to automatic deletion, manual deletion can be undertaken by the user, either in the mobile app or in the web application through the user data access service (*Innsyn*).

3.2.3.2 Registration

Following mobile app installation, users are prompted to undertake SMS verification. Users are onboarded by registering their phone number and authenticating their identities with a B2C login in the app. User profiles and corresponding devices are managed by Azure AD B2C and IoT Hub, respectively.

User accounts live in Azure AD B2C, a different tenant from the main deployment for `dev` and `prod`. This is where phone numbers (as users) and device IDs (as groups) and their associations (group membership) are recorded. The B2C HTML templates are uploaded with Terraform. The B2C custom policy files, which specify how apps can be authenticated, are uploaded via the Azure B2C Identity Experience Framework. The policy files are different for `dev` and `prod` (only in some allowed URLs and tenant IDs) and are uploaded via the upload custom policy link on the Identify Experience Framework page. Login is disabled for `prod`.

3.2.3.3 Data upload

After registration with a phone number and SMS verification, the user receives a unique `ConnectionDeviceID` from the Azure IoT Hub registry. This ID is later used to identify the user whenever data are uploaded, and it identifies data from a specific user in the database. When the mobile app uploads data, it identifies itself using the `ConnectionDeviceID` and then sends the payload, which consists of a JSON document with GPS and BT events to the IoT Hub HTTP endpoint.

3.2.3.4 Deletion

The Smittestopp privacy policy indicates that databases that contain private information, such as phone numbers and GPS (location) data, must be deletable upon request by end users. The policy for deletion can be summarised as follows:

1. All GPS (location) and BT (contact) data will be deleted automatically, 30 days after upload.
2. A user can ask for their data to be deleted at any time.
3. A user who is inactive for more than seven days will be deleted, including all of the user's data.

Deletion according to item 1 is performed by a stored procedure in the SQL Database that runs every night and deletes all data older than 30 days and any other data marked for deletion.

User-initiated deletion according to item 2 is slightly more complex, since it involves deletion from both the B2C and SQL Database. When the user presses the Delete button in the mobile app, the user's device IDs are immediately dissociated from the user's phone number, the user's phone number is deleted from AD B2C, and all device IDs are marked for deletion in AD B2C. These device IDs are immediately unregistered from IoT Hub. At this point, the user's GPS and BT data still remain in the SQL Database, but cannot be linked to a phone number and thus cannot be returned via the internal or external APIs, which operate solely based on phone numbers (*Varslingsløsning, Innsyn*). The remaining data will be deleted as part of the nightly run of the stored procedure in SQL. To avoid the delayed import of data associated with deleted device IDs, after a device ID has been shown to have no data in the SQL Database for multiple days, it is finally removed from AD B2C.

Deletion according to item 3 is the most complex, since IoT Hub does not reliably provide information about the last activity from a device. This issue is resolved by checking the last time a user wrote data in the SQL Database and marking the user for deletion if the user has not written data in the last seven days. This procedure was later updated by introducing a heartbeat from the mobile app to the backend, which can be used to check the time of last activity.

The SQL Database (but not B2C) is backed up using standard cloud database backup strategies. According to item 1, backups must expire in at most 30 days. Data deleted according to items 2 or 3 are not deleted immediately from backups. However, the phone numbers are not backed up and cannot be restored, only anonymous device ID data can, which cannot be extracted from the system except by the database administrators. No database backup restoration occurs during Smittestopp's operation, but the described protection against delayed import also protects against temporarily restoring data from backups. If a backup restores data from a deleted device ID, the deletion procedure will still be running and it will be deleted again upon the next deletion procedure, within 24 hours. This means that, to preserve the deletion policy, database backups older than the deletion recheck window (between two and seven days, according to the configuration) cannot be restored. In summary,

1. Previously deleted data restored from a backup can never be accessed via APIs because it cannot be reassociated with a phone number.
2. Previously deleted data will be deleted again immediately upon the next nightly deletion procedure, since the device ID will still be registered for deletion.

3.2.4 Cloud operations

We now refer to the central part of Figure 3.5, consisting of the Smittestopp backend operations taking place in the Azure cloud. Core components 2 – 7 are involved.

3.2.4.1 Data ingest and storage

Data ingest and storage refers to the operations of components 2 and 3, where the data uploaded by mobile app clients are ingested by the Smittestopp backend. Mobile app clients send event data to the public IoT Hub endpoint, authenticating by certificate. Event data in IoT Hub is considered a *Message*, and IoT Hub adds to each message payload a section with IoT Hub metadata, including the `ConnectionDeviceID` which identifies each unique device and will be used in the database will be used as the device identifier across event data. Event messages are of two types, GPS or BT, with each message containing an array of multiple events.

IoT Hub is configured with four partitions that split incoming data into four parallel flows and a message routing rule that writes all incoming messages into 10 MB uncompressed chunks to Azure Data Lake Storage. In addition, for the first weeks, a streaming endpoint was configured in IoT Hub to which Azure Stream Analytics subscribed for real-time events, which was later abandoned for a pure Data Lake Storage–oriented architecture (see Section 3.2.2.6 for details).[2]

In the Azure Data Lake, filenames are created according to the pattern `yyyy/mm/dd/hh/yyyy-mm-dd-hh-mm-partitionID.json`, where each file corresponds to one 10MB chunk. This naming scheme supports up to four files per minute when including a `partitionID` from zero to three. If the injection rate leads to the creation of more than four files per minute, a sequence number is added to the

[2] Stream Analytics was configured with an input from IoT Hub and outputs for each staging table in the SQL Database. The Stream Analytics Query Language (SAQL) query undertaking the transformation from input to output, mapped each field from the input JSON to the output database table columns, validating the datatype, length, and filters for special characters. In addition, it performed a streaming aggregation on each event array using the `GetArrayElement` function within the execution context of the defined time window.

filename. Data Lake is configured to delete data older than seven days. At peak load, the Smittestopp data lake had up to about 1, 500 individual 10 MB files an hour.

3.2.4.2 SQL database operations

Schema: Table 3.2 presents a list of tables common for the `dev` and `prod` environments.

Table	**Description**
`dbo.agg_gpsevents`	Aggregated table
`dbo.btevents`	BT pairing events
`dbo.dluserdatastaging`	Device information
`dbo.gpsevents`	GPS events
`dbo.grunnkrets`	Geospatial lookups using the geography datatype
`dbo.uuid_activity`	For tracking users who were inactive for 7+ days
`dbo.uuid_id`	Supporting table for Universally Unique Identifier (UUID) <-> internal ID mapping

Table 3.2: Smittestopp main SQL database tables.

Import: Every hour, Azure Data Factory triggers an import job that imports data from Azure Data Lake to the SQL Database. This job is set up as a pipeline with the following steps:

1. Four parallel tasks (`ADSLIMPORTER`) import files from each of the four partitions from the last hour into the table `DLUSERDATASTAGING` in the SQL Database. The import procedure extracts `UUID`, `platform`, `appversion`, `osversion` and `model` from the JSON document and stores them in individual columns. The event payload is stored as an unprocessed JSON in a separate column. The name of the source file is included in the table so that all data can be traced back to the file of origin. Each tasks is authenticated using SQL stored credentials mapped to an Azure service principal with the exact permissions to execute the import task.

2. When `ADSLIMPORTER` is done, another two parallel tasks start. These are the `BTIMPORTER` and `GPSIMPORTER` procedures for importing BT and GPS events, respectively. These procedures process the information imported by `ADSLIMPORTER` in Step 1, as follows:

   a. `GPSIMPORTER` imports all the GPS data from the unprocessed JSON column to the `GPSEVENTS` table.

   b. `BTIMPORTER` updates the `UUID_ID` table so that UUIDs are mapped to internal keys and imports all the BT data from the unprocessed JSON column to the `BTEVENTS` table. Then it updates the `UUID_ACTIVITY` table, which stores the time of last activity for all UUIDs. This is used by the deletion routines to delete data from devices that have been inactive for more than seven days.

3. The last step is the execution of the `AGGREGATOR` procedure to populate the `AGG_GPSEVENTS` table according to the following criteria:

   a. Downsample the GPS events to only include one GPS event every 10 seconds for each UUID.
   b. Execute the `STWITHIN` procedure to carry out a geospatial lookup of the correct `grunnkrets`[3] for each GPS event.
   c. Round all GPS coordinates to two decimal places (both latitude and longitude).
   d. Remove duplicates.

Procedures and functions: The SQL Database has stored procedures and functions for managing access to user data. These allow database users (e.g. AKS) to retrieve processed versions of the raw data, since the database tables cannot be queried directly.

3.2.4.3 Analytics pipeline

The Smittestopp analytics pipeline is executed by running the Dockerised analytics service deployed on AKS (component 7 in Figure 3.5).

Configuration: The list of configurations for the pipeline can be passed as a JSON file to the Docker container, or as environment variables. For instance, the Smittestopp backend has support for setting OSM endpoints using environment variables (`CORONA_OVERPASS_ENDPOINT` and `CORONA_NOMINATIM_ENDPOINT`). DNS names, instead of IP addresses are used.

Database access: Credentials for connecting to the SQL Database are provided as configuration parameters. Since the database can be accessed only from within a VPN, the analytics service can only be run from within the Smittestopp cloud infrastructure (e.g. Analytics VMs, denoted by 6 in Figure 3.5). Data in the SQL Database can only be accessed through predefined procedures and functions, as described in Section 3.2.4.2.

More details about the Smittestopp analytics pipeline are provided in Chapters 4 through 7.

3.2.5 Interactions with web applications

The general goals of the Smittestopp system listed in Section 3.1 and the particular functionalities required of the Smittestopp backend listed in Section 3.2.1 result in three services that access the Smittestopp APIs: contact tracing and notification

[3] The *grunnkrets* is the basic statistical unit defined by Statistics Norway [26].

(*Varslingsløsning*), aggregated statistics (*Kunnskapsinnhenting*), and user data access (*Innsyn*). These services are described below, while the APIs endpoints are listed in Section 3.2.2.10.

3.2.5.1 Contact tracing and notification service (*Varslingsløsning*)

Contact tracing with Smittestopp is performed by NIPH rather than by individuals, as is the case for solutions based on Google/Apple Exposure Notifications (GAEN). With Smittestopp, an employee at NIPH regularly receives a list of infected individuals from the Norwegian Surveillance System for Communicable Diseases (MSIS), which includes their name, Social Security number, and phone number. The Smittestopp API (see Section 3.2.2.10) is then used to look up close contacts for each individual, based on the phone number provided. The lookup process works as follows:

1. NIPH makes a call to the `lookup phone number` endpoint with the phone number of a person who has tested positive for COVID-19 as input.
2. Based on the request, an analysis job is scheduled by creating a record in a Redis database.
3. A link to an endpoint for receiving the result is returned to the caller. This endpoint is called by the NIPH web application until it receives response `200`, which indicates that the request is complete.
4. The analytics service picks jobs from the queue in the Redis database and performs a contact analysis.
5. When the analytics job is complete, the result (list of contacts) is stored in JSON format in the Redis object under the result key.
6. The analysis report is returned to the NIPH web application (see Step 3). From the report, NIPH can notify those who are likely to have been in contact with the infected individual during the period specified in the request.

The actual notification of those who have been in close contact with an infected individual is carried out manually, where a human verifies that all the data are correct and then sends a text message to everyone who have been exposed to infection. The text message warns about potential infection and informs the individual about recommended steps for testing. The long-term plan was to eliminate the manual steps and automate lookup and notification, but this work was never completed. This service does not have any publicly exposed endpoints.

3.2.5.2 Aggregated statistics service (*Kunnskapsinnhenting*)

To support mathematical modelling and epidemiological studies, a service for extracting statistics regarding the pandemic from the Smittestopp data was in development, but never fully implemented. The idea was to provide statistics based on the collected data, such as where infections are occurring (e.g. regions of Norway, different POIs, home vs. work), the current infection rate (increasing, constant, or decreasing), and the average number of close contacts per person. See Chapter 7 for more details.

3.2.5.3 User data access service (*Innsyn*)

For Smittestopp to be compliant with the General Data Protection Regulation (GDPR), it must provide a way for individuals to access any of their personal data used in any way by the system. Therefore, a right of access service is a mandatory part of the Smittestopp architecture. This is implemented as a web service where Smittestopp users can log in and browse all of their GPS-based location data stored in Smittestopp by date and time, using the `egress` call (see Section 3.2.2.10). Furthermore, users can request the access log showing the details of all persons who have viewed their data, including the legal means for doing so. The service was part of `Helsenorge.no` and users where authenticated using ID-porten, a common login solution for public services in Norway. The right of access service, as described above, was never fully operational due to performance issues. An asynchronous solution where a user requests data for later delivery was planned but never implemented.

3.3 Experience: Challenges and lessons learned

In this section, we focus on our experiences with the development and implementation of the Smittestopp backend solution, and describe our lessons learned, along with general insights.

3.3.1 Distributed versus centralised architecture

As indicated in Section 3.1, the Smittestopp backend had a design requirement to run in a centralised manner, relying on cloud components and operations such as a centrally managed data storage, server-side execution of an automated analytics pipeline, and the externally triggered generation of contact and statistics reports by public authorities. This requirement was in line with the intended twofold purpose of the overall system, that is, simultaneous contact tracing and aggregated statistics generation.

During the project's development and launch, as well as after its recall, there were many discussions, both internal and external, regarding the viability of a distributed solution (as opposed to a centralised one). Shortly before the launch of the Smittestopp system, the GAEN initiative demonstrated that a distributed, purely BT-based approach could be used for individualised contact tracing, potentially serving as a more privacy-preserving and secure alternative to centralised solutions.

However, with respect to the requirements mentioned in Section 3.1, storing data only locally on end user devices, in a fully distributed system, could be highly impractical for a number of reasons. Including concerns regarding the data volume, processing time, and network transactions, two fundamental global challenges were identified: the impossibility of listing second-level contacts and the impossibility of generating nationwide aggregated statistics.

In a distributed scheme, an end user device can, based on its location, create a list of devices with which it has had direct contact (along with the corresponding location of each contact). However, listing second-level contacts would require more information located on other devices. Second, generating aggregated statistics based on queries such as 'list all contacts that have happened yesterday at shopping mall *X*' would be a huge and costly operation, requiring communication with all active devices in the system and requesting information on whether they were in the given location at the given time. Therefore, after evaluating these challenges together with the initial system requirements and the need for fast deployment, the Smittestopp backend development was continued in pursuit of a centralised solution.

The security and privacy assessments regarding a centralised solution were also supported by the Ministry of Health and Care Services [17], and it was pointed out that the management of personal information was subject to *Personvernforordningen* articles 6 and 9, such that the overall gain from the system outweighed the potential privacy concerns.

Considering other contact tracing solutions, we see that, as of May 2020, 16 countries [18] had launched or had in development a system based on a centralised approach, including COVIDSafe (Australia), StopCovid (France), Trace Together (Singapore), and HaMagen (Israel), whereas 25 countries used systems based on a decentralised approach, including COVID Alert (Canada), Ketju (Finland), Corona-Warn-App (Germany), and NHS COVID-19 App (UK).[4] With the launch of GAEN, many countries decided to abandon their own solutions in favour of a GAEN-based approach. For several countries, such as the United Kingdom, this meant switching from a centralised to a decentralised solution.

In Norway, discussions were held regarding the adoption of a GAEN-based approach for contact tracing, along with the development a separate solution for generating nationwide aggregated statistics, thereby splitting the desired functionalities into two distinct systems [20, 2].

Mobile apps for digital contact tracing have also been discussed in the previous chapter, under Section 2.2. We discuss privacy- and security-related aspects in Section 3.3.4.

[4] For a comprehensive list of contact tracing solutions around the globe, readers are referred to [22].

3.3.2 Data processing

As described in Section 3.2.1, multiple functionalities were required from the Smittestopp backend with respect to data processing. Each of these imposed different demands on the backend solution and gave rise to various challenges. Below, we focus on the functionalities addressed by the contact tracing and notification service (*Varslingsløsning*) and the aggregated statistics service (*Kunnskapsinnhenting*) as particular examples and discuss some of the data processing challenges we have addressed, along with our experiences.

Functionality 1 (contact tracing using GPS data): The first functionality aims to find those who have potentially been in contact with infected individuals, thus having a risk of being infected themselves. Assuming accurate GPS positions, finding who has been in contact with whom at any given point is supposedly an easy task. The task translates to finding the trajectory of an infected individual and then finding others whose trajectories intersected with this trajectory, that is, those who were at the same location at the same point in time. As shown in Figure 3.6, this means following the red trajectory (infected person) and finding all positions where it intersects with another trajectory, within an allowed distance threshold, such as the blue trajectory (of other person). A very simplified pseudocode, checking everything in a straightforward manner and aiming to solve the intersection problem solely by an SQL query is shown in Figure 3.7, where `infected` and `others` represent tables including GPS data.

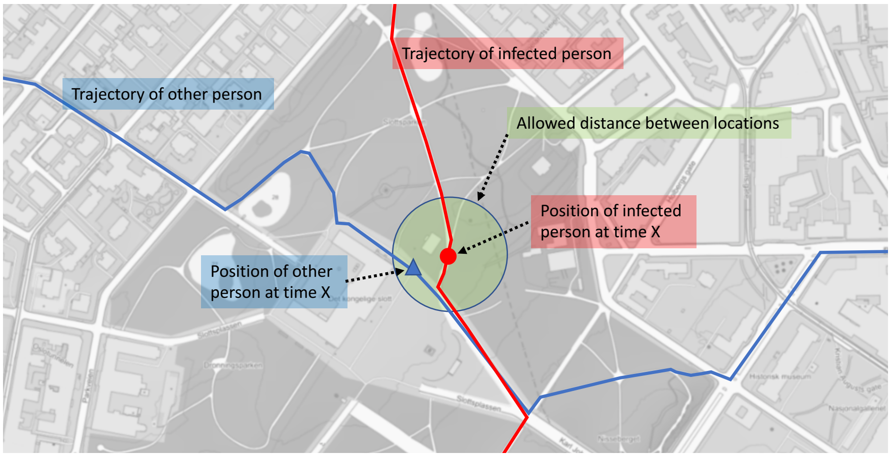

Trajectory of infected person
Trajectory of other person
Allowed distance between locations
Position of infected person at time X
Position of other person at time X

Fig. 3.6: Identifying GPS intersections: for every point on the trajectory of the infected person, find the matching positions of other persons within a distance (radius) X. Map from https://www.norgeskart.no.

Functionality 2 (deriving aggregated statistics): The second functionality aims at generating overall population statistics on how people move and how the disease

```
SELECT (relevant information from) others
FROM infected JOIN others ON
    infected.id = <ID of infected person> AND
    infected.id != others.id AND
WHERE
    infected.location WITHIN (others.location + <allowed distance>) AND
    infected.time BETWEEN <date_from> AND <date_to> AND
    others.time BETWEEN <date_from> AND <date_to> AND
    infected.time OVERLAP others.time
```

Fig. 3.7: Simplified query to find GPS intersections, given the `ID of infected person`, a threshold for the `allowed distance`, and a time range in the form of `date_from` and `data_to`.

spreads. As an example of aggregated statistics, let us assume that we need to find all encounters that happened within a given area, defined by a polygon, corresponding to a Norwegian geographical area unit called *grunnkrets* (the basic statistical unit in Norway, providing a stable and coherent geographical identification). To give an idea of the complexity involved, Figure 3.8 shows how the city of Oslo is divided into *delbydeler*, the statistical unit above the *grunnkrets*, each consisting of several *grunnkretser*. In total, there are about 14, 000 *grunnkretser* in Norway. Thus, for each *grunnkrets*, we should be able to find all BT pairings that happened within the polygon defining this area and can match the time of the contact to the requested time interval (Figure 3.9).

Overview of challenges: Given the simplicity of the pseudocode in Figures 3.7 and 3.9, the above functionalities can intuitively be deemed relatively easy to compute. However, various challenges and complicating factors arise, even in the `dev` database containing a few hundred thousand entries, which is very small compared to the `prod` database in the actual production system containing millions of data records. Below, we discuss these challenges and complicating factors.

3.3.2.1 Date ranges and columnstore storage

The Smittestopp database contains entries from a relatively long time interval (30 days, after the implementation of periodic data deletion as described in Section 3.2.3.4). Storing large amounts of data in one big table generally constitutes an overhead, and, depending on how the data are stored, one might also need to access a large number of disk blocks. However, searching for data within a limited period requires us to access only a small portion of the entries. Bearing in mind that contact tracing typically requires going back only a few days (14 or, more commonly, seven days) and that our queries usually have a granularity of days, we implemented an optimisation in the form of *columnstore* storage, to increase efficiency. Columnstore indexing provides the physical storage of data that is already grouped by day. The speed of our queries can thus be increased.

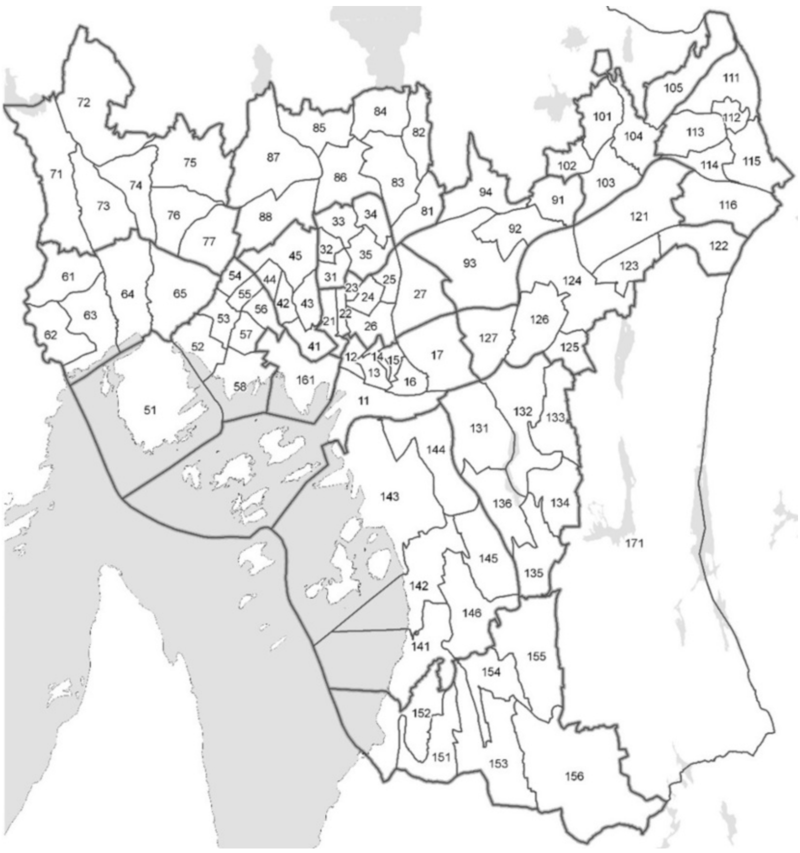


Fig. 3.8: Norway has many *grunnkretser*. The image above shows how Oslo alone is divided into *delbydeler*, each consisting of multiple *grunnkretser* (map from https://no.wikipedia.org/wiki/Delbydeler_i_Oslo).

```
SELECT (encounter data from both tables)
FROM gps-events a JOIN bt-events b ON
    a.uuid = b.uuid AND
    b.pairedtime BETWEEN a.timefrom AND a.timeto
WHERE
    a.location WITHIN <polygon>
    b.pairedtime BETWEEN <date_from> AND <date_to>
```

Fig. 3.9: Simplified query combining GPS and BT data to find all encounters inside a `polygon` within a time range given by `date_from` and `date_to`.

3.3.2.2 Inaccurate GPS measurements

As a technology, GPS itself has limited accuracy [24]. This is further complicated by variations in the location accuracy as reported by different end user device platform and models (see Section 2.6.2 for a discussion of location services in Android and iOS). Thus, we cannot test for exact position matches in our queries and must allow for slack. This is shown in Figure 3.6, where we search for a match within a certain diameter around a given point. Overall, not being able to check for position equality but, rather, for inclusion within a given area, our queries become more complex.

3.3.2.3 Timestamp matching

Clock synchronisation: In a perfect world, all devices would keep the exact same time. However, in practice, different clocks can be off by up to several seconds, which must be considered when we query for matching timestamps. Thus, matching the time between two entries translates to checking whether their timestamps overlap by more than a certain threshold.

Encounter duration: From a contact tracing viewpoint, an encounter must exceed a certain duration threshold to be counted as a valid contact. For GPS data, this threshold was defined as 15 minutes, meaning that an additional check had to be performed to see if the sum of the individual encounters between devices D1 and D2 lasted more than this threshold, before declaring that D1 and D2 had a contact.

Timestamps from location entries: A challenge related to timekeeping in terms of the GPS entries is that the Smittestopp mobile app can only register GPS location information (latitude and longitude) in the form of events with `timefrom` and `timeto` fields, which are not necessarily the same at all times. When calculating encounter durations, the following approach is taken. A contact is reported only if the total encounter duration exceeds the threshold mentioned above.

- If a device is staying in the same place, and the Smittestopp mobile app is reading many GPS events with the same location but different timestamps, the app itself will merge the records, keeping the location and the `timefrom` timestamp from the first data record and the `timeto` timestamp from the last data record in the series. Thus, the difference between the `timeto` and `timefrom` fields can be used to indicate the total time the device has spent in the given location and can be directly added to the total encounter duration.

- If the device is moving, the GPS events read by the app will have `timefrom=timeto`, with different locations indicated by consecutive records. Thus, if a device is moving through points P1 and P2, which are identified as part of an intersecting path (two devices are moving together on this path), the time between the `timefrom` timestamp of point P1 and the `timefrom` timestamp of the consecutive point P2 can be added to the total encounter duration.

3.3.2.4 Calculating speed and distance

From an analytics perspective, trajectory calculation also involves determining the mode of transport for nonstationary devices (see Chapter 4 for details on the

Smittestopp analytics pipeline). However, in order to establish a mode of transport, information about the speed is necessary. Various reports indicate that the estimated speed reported by the built-in GPS of end user devices such as smartphones, tablets, and smartwatches are highly inaccurate. Therefore, we needed to calculate the speed of the moving devices ourselves, based on the location and timestamps indicated by consecutive GPS records (using the `lag` function).

3.3.2.5 Calculating distance on a sphere

Due to the curvature of the earth, using the Pythagorean theorem [31] to calculate the distance between two pairs of coordinates is not 100% accurate. As a remedy, the Haversine distance [27] can be used to find the great circle distance between two points on a sphere, given their longitudes and latitudes. However, computing the Haversine distance is a complex and costly operation. In the Smittestopp backend, we implemented a simplified version of this distance as a trade-off between accuracy and processing costs. The simplification produces minimal errors, since, for our use case, the distances involved are relatively short.

Given that the diameter of the Earth is 12, 742, 016 metres, the distance between two points can be calculated as follows:

```
distance between points A (@LatA, @LongA) and B (@LatB, @LongB) =
   12742016 * asin(min(sqrt((sin(radians(@LatB - @LatA)/2)
   * sin(radians(@LatB - @LatA)/2)
   + cos(radians(@LatA))
   * cos(radians(@LatB))
   * sin(radians(@LongB - @LongA)/2)
   * sin(radians(@LongB - @LongA)/2)))))
```

This trade-off formula provides sufficiently accurate distances, but it is a question of whether the computation is still overkill. Most of the distances we calculate are rather short (with only a small error due to the Earth's curvature), and the accuracy of the GPS records themselves are also questionable. Thus, in a future system similar to Smittestopp, the differences between a plain Pythagorean calculation and a Haversine calculation, in terms of accuracy versus computing costs, should be investigated more closely.

3.3.2.6 Location pre-filtering

To reduce the number of entries in the `JOIN` operation when we search for matches within an area, we can pre-filter entries, that is, group them per unit bounding box of 1 kilometre, with respect to their latitude and longitude.

The distance between two consecutive degrees of latitude is constant (111.045 kilometres everywhere on Earth):

```
latitude between <latitude> ± (1 / 111.045)
```

The distance between two consecutive degrees of longitude is not constant (the distances are smaller the further away they are from the equator):

```
longitude between <longitude> ± (1 / (111.045 * cos(radians(latitude)))
```

3.3.2.7 Trajectory segmentation

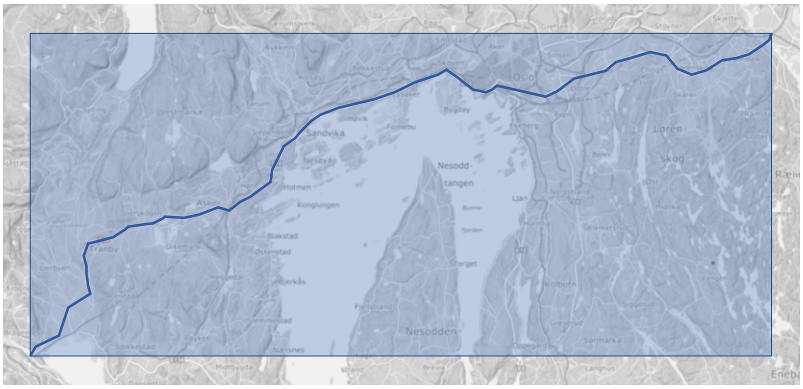
Taking into account all the features of our database, it was clear from the start that the queries from Figures 3.7 and 3.9 might not be straightforward. However, tuning such queries on the `dev` database is still possible with return times in seconds or minutes, that is, seemingly within the operational thresholds of a normal web application. On the other hand, going from the `dev` database to the `prod` database (number of database records on the order of one hundred million, versus the meagre one hundred thousand in `dev`) leads to new challenges.

On the largest instance on Azure, with 80 vCores, such queries took days to finish, such that we could not depend on the database to run queries that were too complex over too large a time span over too large a geographical area. This demonstrated the need for a distributed computing approach, where we would divide our tasks into smaller subtasks, and assign these to different processors or machines.

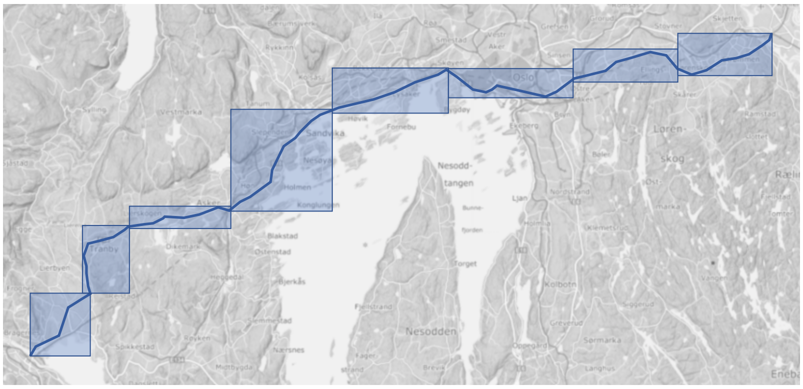
After several optimisations were tested, the final deployed solution would divide a user's trajectory into smaller segments of geographical areas (bounding boxes) and time intervals, as depicted in Figure 3.10. To offload the database server, user data from these bounding boxes and time intervals would be sent to the analytics pipeline (described in Chapter 6) for further fine-grained processing.

3.3.2.8 Data sanity checks

Importing data from millions of devices outside one's control naturally resulted in a certain number of invalid data samples in our database. Among the samples collected, we observed both erroneous timestamps (date and/or time) and erroneous GPS locations. For instance, numerous records had timestamps indicating dates long before or after the period the Smittestopp system had been running. There could have been several reasons for this, but since we had no way of knowing the correct timestamps, such records were removed. Additionally, we also observed anomalous latitude and longitude values, where devices known to be in Norway had sent GPS records indicating locations far outside the country. Again, there could have been different reasons, such as hardware or operating system problems, GPS jammers in certain areas, and so forth. Nevertheless, these records were also removed from the database. To handle such cases, data sanity checks during data import were used, and sanity check operations were added to the mobile app at a later stage, to avoid sending erroneous data in the first place.



(a) Searching an entire trajectory within one operation, yielding a huge area over an entire period.



(b) Dividing the trajectory into multiple smaller segments, greatly reducing the search area, with a shorter, distinct time interval for each segment.

Fig. 3.10: Sample search area when an infected individual travels from Drammen to Lillestrøm.

3.3.2.9 Database schema updates

To optimise certain data operations and calculations that were run multiple times within the analytics pipeline, we traded off storage for faster data analysis. For instance, we introduced the notion of precalculated values during the data import stage. For a given device, these include the speed and distance between the current and previous locations, as well as an associated *grunnkrets* for every GPS entry, depending on the latitude and longitude. To store these values, database tables were augmented with additional columns and new tables were constructed to optimise operations for various specialised queries – to a large degree breaking traditional database normalisation rules, but yielding considerable gains in terms of processing speed.

3.3.2.10 Moving operations to end user devices

Another way of distributing the processing load is to move some of the pre-calculations and data sanity checks to the end user devices, which collectively constitute a far more powerful computing source than any of our machines in Azure (which were 1.5 million multi-core devices). For example, a device's speed and distance between two points could easily be added to the computation in the mobile app before data are sent to the IoT Hub, avoiding frequently repeated and computationally expensive operations on the server side. The same could be done with data sanity checks. In the final stages of the project, with the system being shut down, these tasks were delayed and now remain as future suggestions.

3.3.2.11 Manual versus automatic tracing

There is no doubt that digital solutions allowing for the automatic and large-scale execution of certain analyses can greatly assist organisations such as NIPH with their efforts in contact tracing. However, it is important to note that the quality of an analysis is never better than the dataset on which it is based. In the previous sections, we pointed out several sources for low-quality data, including but not limited to timestamp errors or inaccuracy in GPS and BT samples. Some of the challenges associated with low-quality data can be alleviated with certain system optimisations, and some cannot. Therefore, the analyses from systems such as Smittestopp should not be assumed to be completely error-free. There is also a greater, more fundamental challenge for such solutions: people can turn off their devices or disable contact tracing apps, meaning there are no data at all.

There were several occasions during the testing period when Simula was informed by NIPH that the automatic system had detected more encounters than the manual tracing process, which was encouraging. However, digital contact tracing solutions should not be perceived as complete alternatives for replacing manual procedures, but, rather, as supplementary mechanisms to support the existing systems. For a more detailed discussion of digital versus manual tracing, see Chapter 6.

3.3.3 Cloud optimisations

Some of the challenges we faced through the development of the Smittestopp backend concern the efficient use of the cloud components described in Section 3.2.2. Below, we provide examples for two such aspects.

3.3.3.1 Handling data import

As mentioned in Section 3.2.4.1, the Azure Stream Analytics service was used in the first weeks of development for processing incoming events, for basic validation, filtration, and pre-processing, and for pushing data to the SQL Database. The service was later abandoned for a pure Data Lake Storage–oriented architecture with batch ingestion.

One of the biggest challenges causing this change was that the Smittestopp mobile app instances on some smartphones were sending the same event data over and over again, while others were sending data at very high frequencies (up to one event per second), which needed to be filtered out early in the data processing pipeline.[5] These filters required database lookups that are better to implement in set-oriented batch processes. The shift from the Azure Stream Analytics service also improved the troubleshooting and end-to-end tracking of individual device events overall.

3.3.3.2 Managing load on components

One example related to managing the operational load on the backend components was the need to optimise the use of the OSM servers and, more specifically, to prevent the Overpass API from becoming a bottleneck during the execution of the Smittestopp analytics pipeline.

The background to this problem is that we were seeing a bottleneck in the calls to this server while querying POIs for a given trajectory. The initial implementation tried to utilise the server better by sending many small requests in parallel, but this was perceived as causing a bottleneck. To address this problem, we introduced an alternative code path in the analytics service (AKS) that tries to exploit the fact that the bounding boxes for the points along a trajectory will have considerable overlap. This way, for a typical trajectory, the number of requests were reduced to one. Further work would still be needed to handle very long trajectories.

3.3.4 Ethical, privacy and security aspects

The use of GPS and BT data in association with user IDs carries the risk of revealing personal information. However, one of the goals of the Smittestopp backend is to inform persons if they have been in contact with an infected individual, necessitating an ID in one form or another to be stored in the system. Therefore, a number of measures were implemented in the Smittestopp backend to make the system as

[5] On the mobile app side, bugs in the development of the software could lead to the same data being sent over and over again, whereas the frequency of sending data had inherent limitations and differences with respect to different platforms (Android vs. iOS). From an analytics perspective, it was also a matter of discussion *how* the local data on the smartphones should be pre-processed before being sent to the database in the backend.

secure and privacy preserving as possible. In this section, we elaborate on some of these measures and associated challenges.

3.3.4.1 Overview of security measures

The Smittestopp backend relies on the Microsoft Azure cloud infrastructure at an enterprise service tier, thereby inheriting all generic security measures available for business solutions. All Azure services employed by the Smittestopp backend use the Azure AD B2C enterprise identity service to provide single sign-on and multi-factor authentication.

Data in transit: The Smittestopp backend uses encrypted communications in all mobile app and web interactions (see Sections 3.2.3 and 3.2.5), firewalls in authentication processes and hidden APIs (see Section 3.2.2.10), and HTTPS access to all servers with TLS certificates (see Section 3.2.2.7).

Data at rest: All storage in Azure SQL Database, storage containers, and so on, as well as the respective backups are secured with encryption at rest, and the encryption keys are stored in the customer key vault service. The backend also employs multilevel access, limited user groups (depending on the level of authorisation), and cross-organisation data splitting (according to the scope of authority).

3.3.4.2 Data anonymisation

Data anonymisation refers to the removal of personally identifiable information from data sets, so that the individuals the data describes remain anonymous [23]. Within the Smittestopp system, all operations after the registration are based on devices (not users) and are undertaken through the use of non–human-readable UUIDs.

In the Smittestopp backend, with the exception of AD (component 1 in Figure 3.5), none of the components use any identifier that can be associated with an individual or phone number. All data operations within the backend (components 2 – 8 in Figure 3.5, including IoT Hub, Data Lake, and SQL Database) use the device handle UUID, as mentioned above. AD lookups in the `prod` environment are subject to extremely tight restrictions and strict auditing, with no more than two people in the entire project having access to the directory.

Although it is not possible to run a completely privacy-preserving operation within the centralised confines of our operation (see Sections 3.1 and 3.2.1 to review the design specifications, and Section 3.3.1 for a general discussion of centralised vs. distributed architectures), a certain level of privacy has been ensured in this regard. We refer readers to Chapter 7 for a detailed discussion of anonymity within the context of aggregated statistics.

3.3.4.3 Data storage

From the start of the project, it was a priority that the physical storage of any user data associated with Smittestopp complied with the jurisdiction of the European GDPR. In this regard, all Smittestopp backend components resided in Europe since the first day. Major components such as Data Lake and SQL Database were located in Norway from the beginning. However, at the time we went into production, Microsoft's data centre in Norway had just recently opened, and a number of Azure services were not yet available in Norway. Some components, such as the IoTHub and Stream Analytics engine, were located in the Northern Europe data centre (Ireland) for a few weeks. Afterwards, in the second phase, the backend was reconfigured, and most of the services were relocated to Norway. During the reconfiguration, we also scrapped Stream Analytics and moved to a data load from file approach, as mentioned in Sections 3.2.2.6 and 3.3.3.1.

3.3.4.4 Data access

Access to the Smittestopp backend operates on a component level, with only a handful of people having access to critical components such as Data Lake and SQL Database (notwithstanding the differentiation between the `dev` and `prod` environments). All access to customer assets in the Azure subscription were controlled and audited by NHN.

Access by Microsoft: The Data Processing Agreement (DPA) between Microsoft and NHN is the judicial framework that Microsoft depends on with respect to GDPR [11]. According to this agreement, Microsoft is contractually obligated to provide sufficient guarantees to meet key requirements of GDPR.

Access by developers: Access to any services related to data storage and data processing were granted to vetted developers by NHN on a subscription basis. Developer access was granted to a limited number of Simula and NIPH personnel.

3.3.4.5 Bluetooth IDs

A weakness in the first release of Smittestopp was the use of static client IDs for tracing BT pairings. The client ID is exchanged with another instance of Smittestopp when two devices are within BT range of each other. With static IDs, it is possible to non-continuously track users through BT beacons, placed at strategic locations, that scan for the presence of specific IDs. To avoid this type of tracking, the client ID should change over time, which can be achieved by rotating IDs. Therefore, an endpoint (see Section 3.2.2.10) for providing rotating client IDs was implemented, and support for rotating client IDs was added to the mobile app. Using this endpoint, the mobile app can request a set of random client IDs and use each ID for a period before it is changed. When the mobile app runs out of IDs, it requests a new set of

IDs from the backend. This feature was implemented and tested, but never released to the general public.

3.3.4.6 Research and development data

The acquisition of data to use for research and development is often challenging in situations involving personal information. According to the sandbox approach described in Section 3.2, we maintained two instances of the backend system, namely, the `dev` and `prod` environments. Here, the `prod` environment was completely out of reach for the majority of the team, and data collected in the `dev` database, contributed by the informed volunteers from Simula and NIPH, were available for development purposes. Although this helped us build our algorithms and prototypes, the fact that we were never able to test at the real scale remained a challenge. Nevertheless, this was an important measure to protect the production user data from potentially unstable development versions of the services and functions.

3.4 Summary and conclusions

In this chapter, we presented an overview of the Smittestopp backend solution, elaborating on its purpose, design, implementation, and operations. We traced the evolution of the solution over time as a response to project goals and requirements, reflected upon various aspects of performance, and, finally, touched upon the open challenges that can hopefully motivate future work. The overall Smittestopp system was deployed for public use in only about give weeks, reaching a record 1.5 million registrations in a short time and collecting hundreds of millions of data records. Despite the issues and challenges that still remain to be open, we hope that our experiences and insights can support future projects of a similar nature, by allowing other teams to learn from our mistakes and use our preliminary results.

References

[1] Vilni Verner Holst Bloch (Statistisk Sestralbyrå). Standard for delområde- og grunnkretsinndeling. https://www.ssb.no/klass/klassifikasjoner/1(inNorwegian),accessedFebruary2021.

[2] Simula Research Laboratory and Simula Metropolitan. Sammenligning av alternative løsninger for digital smittesporing. https://www.simula.no/sites/default/files/sammenligning_alternative_digital_smittesporing.pdf(inNorwegian),accessedFebruary2021.

[3] Microsoft. API management. https://azure.microsoft.com/en-us/services/api-management/,accessedFebruary2021.

[4] Microsoft. Azure IoT Hub documentation. https://docs.microsoft.com/en-us/azure/iot-hub/,accessedFebruary2021.
[5] Microsoft. Azure Kubernetes Service (AKS). https://azure.microsoft.com/en-us/services/kubernetes-service/ ,accessedFebruary2021.
[6] Microsoft. Azure Maps. https://azure.microsoft.com/en-us/services/azure-maps/,accessedFebruary2021.
[7] Microsoft. Azure Monitor. https://azure.microsoft.com/en-us/services/monitor/,accessedFebruary2021.
[8] Microsoft. Hyperscale service tier. https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tier-hyperscale,accessedFebruary2021.
[9] Microsoft. Introduction to Azure Data Lake Storage Gen2. https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction,accessedFebruary2021.
[10] Microsoft. Microsoft Azure. https://azure.microsoft.com/en-us/ ,accessedFebruary2021.
[11] Microsoft. Microsoft's GDPR Commitments to Customers of our Generally Available Enterprise Software Products. https://docs.microsoft.com/en-us/legal/gdpr,accessedFebruary2021.
[12] Microsoft. Welcome to Azure Stream Analytics. https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction,accessedFebruary2021.
[13] Microsoft. What is Azure Active Directory B2C? https://docs.microsoft.com/en-us/azure/active-directory-b2c/overview,accessedFebruary2021.
[14] Microsoft. What is Azure Data Factory? https://docs.microsoft.com/en-us/azure/data-factory/introduction,accessedFebruary2021.
[15] Helse Norge. Smittestopp. https://helsenorge.no/smittestopp, accessedSeptember2020.
[16] norskhelsenett. Nasjonale e-helseløsninger. https://www.nhn.no/ ,accessedFebruary2021.
[17] Helse og omsorgsdepartementet. Forskrift om digital smittesporing og epidemikontroll i anledning utbrudd av Covid-19. https://www.regjeringen.no/contentassets/116076d9a39b473a97d97474048e1fb0/kgl.-res.-27.-mars-digital-smittesporing.pdf(inNorwegian) ,accessedFebruary2021.
[18] Patrick Howell O'Neill, Tate Ryan-Mosley, and Bobbie Johnson. A flood of coronavirus apps are tracking us. Now it's time to keep track of them. https://www.technologyreview.com/2020/05/07/1000961/launching-mittr-covid-tracing-tracker,accessedFebruary2021.
[19] OpenStreetMap. Welcome to OpenStreetMap! https://www.openstreetmap.org/,accessedFebruary2021.

[20] Simula. En ny runde med digital smittesporing? https://www.simula.no/news/en-ny-runde-med-digital-smittesporing(inNorwegian) ,accessedFebruary2021.
[21] Simula. Smittestopp backend. https://github.com/simula/corona/tree/master/backend/doc(authorizationrequiredforaccess) ,accessedFebruary2021.
[22] Wikipedia. COVID-19 apps. https://en.wikipedia.org/wiki/COVID-19_apps,accessedFebruary2021.
[23] Wikipedia. Data anonymization. https://en.wikipedia.org/wiki/Data_anonymization,accessedFebruary2021.
[24] Wikipedia. Global positioning system. https://en.wikipedia.org/wiki/Global_Positioning_System,accessedFebruary2021.
[25] Wikipedia. Google Maps. https://en.wikipedia.org/wiki/Google_Maps,accessedFebruary2021.
[26] Wikipedia. Grunnkretser i Norge. https://no.wikipedia.org/wiki/Grunnkretser_i_Norge,accessedFebruary2021.
[27] Wikipedia. Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula,accessedFebruary2021.
[28] Wikipedia. Nominatim. https://wiki.openstreetmap.org/wiki/Nominatim,accessedFebruary2021.
[29] Wikipedia. Overpass API. https://wiki.openstreetmap.org/wiki/Overpass_API,accessedFebruary2021.
[30] Wikipedia. Point of interest. https://en.wikipedia.org/wiki/Point_of_interest,accessedFebruary2021.
[31] Wikipedia. Pythagorean theorem. https://en.wikipedia.org/wiki/Pythagorean_theorem,accessedFebruary2021.