



Smart Survey Implementation

Grant Agreement Number: 101119594 (2023-NL-SSI)

Work package 3

Developing Smart Data Microservices

Deliverable 3.2: Smart baseline stage report

Version 1.2, 2024-06-27

Prepared by:

Joeri Minnen (hbits, Belgium)
Pieter Beyens (hbits, Belgium)
Ken Peersman (hbits, Belgium)
Enak Cortebееck (hbits, Belgium)
Jonas Klingwort (CBS, Netherlands)
Tim de Jong (CBS, Netherlands)
Tom Oerlemans (CBS, Netherlands)
Jerome Olsen (Destatis, Germany)
Joël Van Hoorde (Destatis, Germany)
Adrian Montag (Destatis, Germany)
Miriam Engel (Destatis, Germany)
Fabrizio De Fausti (ISTAT, Italy)
Claudia De Vitiis (ISTAT, Italy)
Marco Terribili (ISTAT, Italy)
Francesca Inglese (ISTAT, Italy)

e-mail address : Joeri.Minnen@hbits.io

mobile phone : +32 (0)497 189503

Disclaimer: Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Eurostat. Neither the European Union nor the granting authority can be held responsible for them.

Index

Index.....	2
1. General introduction	4
Objectives of WP3	4
Documentation strategy	5
Git repository and used open-source libraries	5
Demonstrations.....	5
2. Timeline	7
3. Microservice software architecture	10
Views	10
Perspectives	16
4. Functional and non-functional requirements of the Receipt Scanning Microservice	18
Business requirements	18
Functional requirements.....	21
Non-functional requirements.....	25
5. OCR microservice (Receipt Scanning Microservice – part 1)	28
Design.....	28
Implementation.....	37
Integration.....	37
Test.....	43
6. COICOP classification microservice (Receipt Scanning Microservice – part 2)	44
Design.....	44
Current state of work and outlook.....	46
Implementation.....	47
Integration.....	47
Test.....	47
7. Functional and non-functional requirements of the GeoService Microservice.....	48
Business requirements	48
Functional requirements.....	50
Non-functional requirements.....	54
8. Geolocation microservice determining stop-track clusters (GeoService Microservice – part 1) .	56
Design.....	56
Implementation.....	58
Integration.....	58

Test.....	58
9. Geolocation microservice detecting the mode of transport (GeoService Microservice – part 1)	59
Design.....	59
Implementation.....	60
Integration.....	61
Test.....	61
10. HETUS classification Microservice (GeoService Microservice – part 2)	62
Design.....	62
Implementation.....	65
Integration.....	65
Test.....	65
11. MOTUS architecture	66
12. CBS architecture	68

1. General introduction

This document serves as the smart baseline report and provides a second overview of the work done in work package 3 of the SSI project.

The main goal of this work package is to develop microservices and to arrive to the overall goal 'to develop, implement and demonstrate the concept of Trusted Smart Surveys, realizing a proof of concept for the complete, end-to-end data collection process and demonstration of a solution'. Work package 3 is situated at the Development level, where the microservices are being developed as platform-independent components.

Three different microservices have been indicated to be developed. Throughout the project it has been decided to create per microservice two underlying microservices, as listed below. Each time a non-domain specific part and a domain specific part is built. The non-domain specific part can be used in various statistical domains, whereas the domain specific part is linked to one specific statistical domain and has often country and language requirements:

- Receipt Scanning Microservice
 - OCR Microservice – non-domain specific
 - COICOP classification Microservice – HBS + country/language restrictions
- GeoService Microservice
 - Geolocation microservice – non-domain specific¹
 - HETUS classification Microservice – TUS + country/language restrictions
- Energy Microservice
 - To be defined

WP3 is responsible to develop the non-domain specific microservices as shareable environments to the SSI/ESS, and to integrate them into the end-to-end data collection process. WP3 has a close relation to WP2 which defines the AI/ML models for WP3 to be (further) developed and integrated into microservices. WP3 also provides support to integrate the microservices with the data collection platforms. After integration WP3 can test the microservices in interaction with the users to close the end-to-end process.

Every microservice can hold various models or algorithms that together combine to a process flow within a microservice. One or more microservices integrated to a data collection platform define a process flow to collect and process statistical data.

Objectives of WP3

The main objectives of WP3 are:

- Develop the selected microservices
- Develop the APIs between the microservices and the core platforms
- Setup and perform development tests to support the development of the microservices and APIs, in an interactive and iterative manner
- Document the microservices and APIs
- Support platforms and NSIs to include the microservices in the core platforms

¹ The underlying models/algorithms to define stop/track clusters and to derive a mode of transport on the track clusters are discussed in a different chapter.

- Perform a pentest
- Perform a stress test (load performance of eg. geolocation data)
- Containerise the microservices
- Describe the architecture of the core platforms
- Describe the architecture of the (developed) microservices
- Describe and execute the deployment strategy for both the core and microservices
- Write PDCA-cycles
- Keep and maintain a public Git repository
- Coordinate with the participating countries
- Provide support to WP1, WP2, WP4 and WP5

Documentation strategy

This report is the result of the writing of the technical documentation. At first the general architecture for microservices is presented. Next the within WP3 (to be) developed microservices are documented.

For each microservice the same approach is followed. First the business, functional and non-functional requirements of each microservice are discussed as they are the blueprint of the work that follows. Every microservice has a non-domain and a domain specific part. Non-domain means that that part can be integrated in a different data collection process than the HBS and TUS process that is envisioned within the SSI project. The second part of the microservice then links to specifically the HBS/COICOP and TUS/HETUS guidelines. Despite this split each part of the microservice has the same subsections: design, implementation, integration and test.

The document ends with providing more information about the data collection platforms to which the microservices are integrated. These platforms are the @CBS platform (CBS) and the MOTUS data collection platform (hbits).

Git repository and used open-source libraries

WP3s output is only partially represented by the written documentation. The goal of WP3 is to develop the three microservices. Therefore, the most impactful output is available in the Git repository that will be made available at the end of the project. This Git repository not only hosts the by WP3 generated code but also contains more and specific information for the NSI that are willing to know more about the microservices on a technical level in order to integrate these environments into their own end-to-end data collection process.

The Git repository is available at [TODO], and entrance during the SSI project is granted upon request.

WP3 by themselves makes use of open-source libraries and AI/ML algorithms, as components and which still need to be configured and trained. In this documentation we will refer to these sources so the reader can easily find this information.

Demonstrations

During the SSI-project multiple demonstrations were given. These demonstrations are recorded and are made available via OpenSocial:

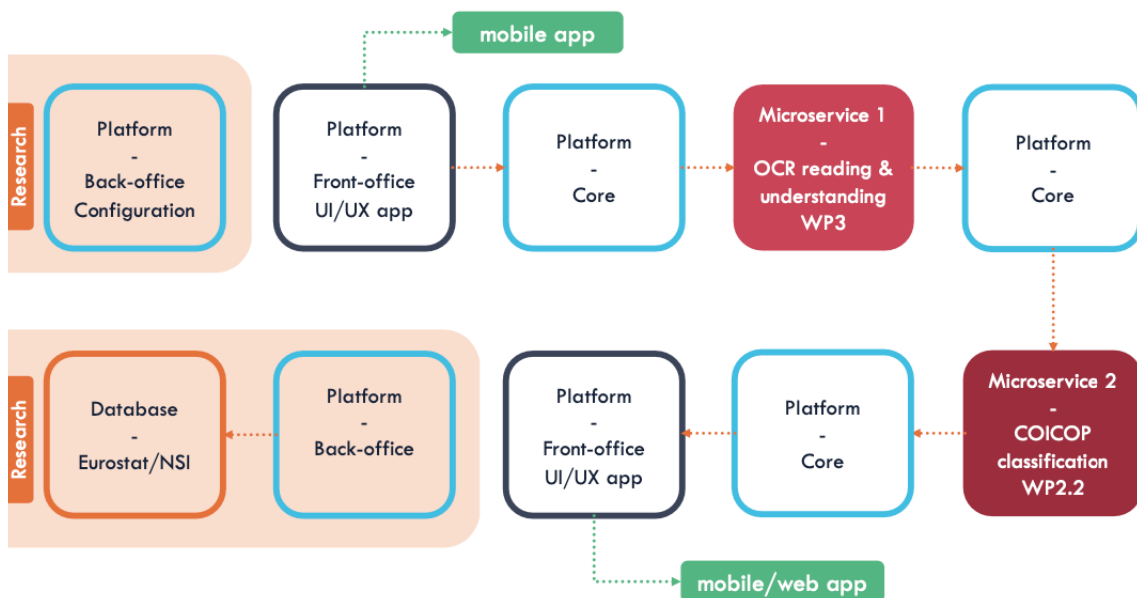
- First Informational meeting SSI on October 20th 2023:

Presentation of the status of three separate models for the OCR microservice. The presentation discusses the strategy to combine the models in a sequence of actions, and to relate/integrate the OCR microservice to the respective platforms.

- Demonstration given by hbits and CBS: <https://cros.ec.europa.eu/book-page/information-session-october-2023>
- PowerPoint: <https://cros.ec.europa.eu/book-page/information-session-october-2023>

- Second Informational meeting on March 22nd 2024:

Presentation of the operational OCR microservice, and the integration of the service to an independent data collection platform, starting from a theoretical end-to-end data flow. Both the OCR and the COICOP microservice are presented. See data flow below.



The presentations accordingly discuss all components, the integration to the platforms and the training strategy.

- Demonstration: <https://cros.ec.europa.eu/book-page/information-session-march-2024-wp3>
 - Demonstration 1 – Integration of OCR microservice given by hbits
 - Demonstration 2 – Training of AI/ML OCR and document understanding model given by Destatis
 - Demonstration 3 – Status of COICOP classification and presentation of search tag strategy given by Destatis and CBS. This demonstration was given in collaboration with WP2 as the responsible work package for the COICOP classification microservice.
- PowerPoint: <https://cros.ec.europa.eu/book-page/information-session-march-2024-wp3>

2. Timeline

Looking to the timeline, the first important task was to develop a general approach to design a microservice software architecture. In doing so the SSI-develop microservices are developed according to a vast set of runtime functional elements that can variate depending on the needs but nonetheless give structure to the development of these microservices. The SSI consortium suggests to the ESS to use this structure for newly build microservices within NSIs. It also holds the suggestion to containerise the microservices as being documented.

The runtime functional elements provide the building blocks for each of the planned microservices. The first (overall; see introduction for the explanation) microservice to be developed is the Receipt Scanning Microservice, followed by the GeoService Microservice and the Energy Microservice.

The work on the Receipt Scanning Microservice (OCR and COICOP microservice together) started in May 2023 with the description of the functional and non-functional requirements. Accordingly, the development of the microservice started. The progress was shown to the consortium by means of two demonstrations. The first demonstration in October 2023 showed 3 separate AI/ML models being part of the OCR microservice. The first model focuses on how a ticket (contour, orientation) has been prepared, the second model deals with how text from a ticket is extracted and placed into position boxes, and the last model connects the boxed text to standardised labels holding relevant elements that (can) appear on a ticket.

The second demonstration in April 2024 showed improvements in all three models and most importantly the developed AI/ML models were bounded in a sequence of actions. This means that upon arriving of a ticket to the microservice the ticket has automatically been processed sequentially model by model and the derived information is made available in the microservice database. Via an API this information becomes available to other environments. This setup also makes it possible to place this microservice in an end-to-end data collection process within an NSI.

It is important to notice that the OCR Microservice has the assumption to be non-language, non-country and (even) non-shop specific. In order to achieve this the separate models have to be trained, and for this a PDCA model has to be specified. The second presentation showed a solution on how to annotate these tickets following a standardised procedure of attaching labels to content on the ticket.

The development work on the OCR Microservice ended March 2024. This intermediary report holds the vast amount of written knowledge that has been created during the SSI project, while the developed code is available in the Git repository. The development of the COICOP classification microservice keeps running in WP2, with support from WP3.

From April 2024 onwards hbits and CBS have the knowledge and code available to integrate the OCR Microservice into their data collection platform, respectively MOTUS and @CBS. The integration is in view of the HBS. This integration should hold the entire process starting from a user/respondent taking or uploading an (e-)ticket via the MOTUS and @CBS app, sending it over to the OCR Microservice in order to be processed through the developed models and where this derived information then is showed as tentative data back into the used application and available to the user/respondent to be edited in order to commit the ticket/data. Depending on the integration

strategy used these steps are supplemented with passings through the platforms' core environments, e.g. to achieve a higher privacy and security in the exchange of data between the environments.

Once the integration is done, and technically tested, WP2 can start to test the integration in small scale tests and large-scale pilots in interaction with the user.

An important output of the OCR Microservice are the product descriptions (next to the price of the purchases and other contexts like some product metrics or reductions). The product descriptions need to be classified to a COICOP code. This connection of the production description to a COICOP code is done by WP2.3 and leads to the development of the COICOP microservice. When this COICOP microservice is functional and well described it can be integrated into the end-to-end process accordingly.

The second microservice is the GeoService Microservice, starting in November 2023 with the same approach as the Receipt Scanning Microservice. First phase is the writing of the documentation on the functional and the non-functional requirements. Next phase is the development of the microservice. Likewise, the Receipt Scanning Microservice the development is split in a non-domain specific (Geolocation Microservice) and a domain specific part (HETUS classification Microservice), where the first part delivers a microservice that can be used in any end-to-end production flow that is willing to make use of geolocation points, and the second part is related to the specific use within the HETUS data collections.

The main objective of the Geolocation Microservice is to derive information on stop points (clusters) and link a transport mode to the track clusters. These stop points are found, after pre-processing, through the use of an AI/ML model that takes into account spatial and temporal parameters. The contextualisation is done by connecting these stop points to a third-party POI/places database, be it OSM or Google places. This work is foreseen to be completed end of June 2024. Again, to be made available to the data collection platforms, the microservice needs to be integrated into the end-to-end process of a data collection platform that holds the productive application which the users/respondents employ to register Time Use Survey data based on the HETUS guidelines. The integration runs until October 2024 and is accordingly used for pilot testing via WP2. To date no demonstration was given, mainly due to a lack of (training) data.

Just as for HBS there is a specific development for TUS. WP2.2 will use the output of the Geolocation Microservice to prognose the motivational context of the transport activity within the HETUS classification Microservice. To achieve this goal, WP2.2 will also incline social and work-related background characteristics in the developed model. To realise this requirement and to achieve privacy and security the data collection platform needs to stand between the Geolocation Microservice and the HETUS classification Microservice.

The last and third microservice is the Energy Microservice. The preliminary meetings have taken place.

A specific topic coming to the surface during the SSI project is the need for (nearly) real-time data processing. In interaction with users, the output of the microservice is presented to the user in order to be edited and submitted as committed data. Therefore, for the SSI, ticket and geolocation

data needs to be performantly processed to output data (variables). In the course of developing AI/ML models it became clear that not all program languages were fitting towards to need of real-time processing (eg. R has a slow performance in comparison to C++).

The underlying report builds upon the previous report, and besides adding new knowledge it also retakes information (as mainly is the case for Chapter 3 and 4). This and new work have been discussed during online meetings and during physical workshops organised by CBS (Heerlen), Destatis (Bonn) and hbits (Brussels). Accordingly, the chapters are reviewed by the work package leaders. After an in-depth review also the countries and the experts related to WP3 have been provided the opportunity to reflect upon the documents.

The work is technical of nature, and is written towards the staff of NSIs that will need/are invited to integrate the microservice into their end-to-end data collection process.

Complementary to this work also belongs the given demonstrations and the available code in the Git repository.

3. Microservice software architecture

This chapter describes a generic architecture for data processing microservices.

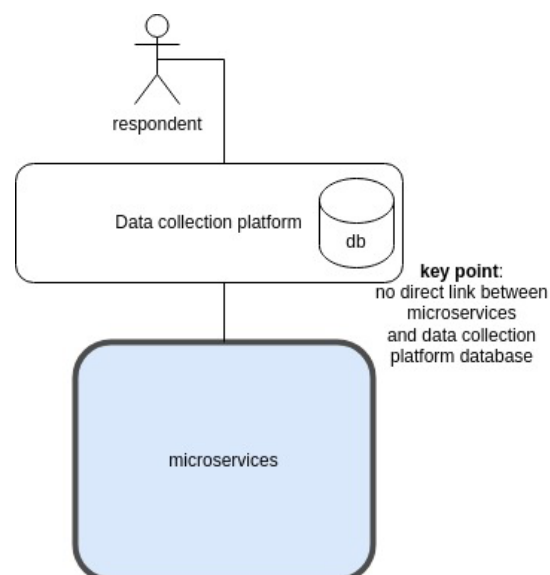
The structure of this chapter is based on views and perspectives. Views illustrate the structural aspects of an architecture (e.g. where is data stored?), while perspectives consider the quality properties (e.g. scalability) of the architecture across a number of views.

A microservice is seen as an independent environment and, in that respect, not coupled to a specific data collection platform.

Views

Context view

The diagram below shows how microservices (as a black box) fit into a general data collection platform architecture.

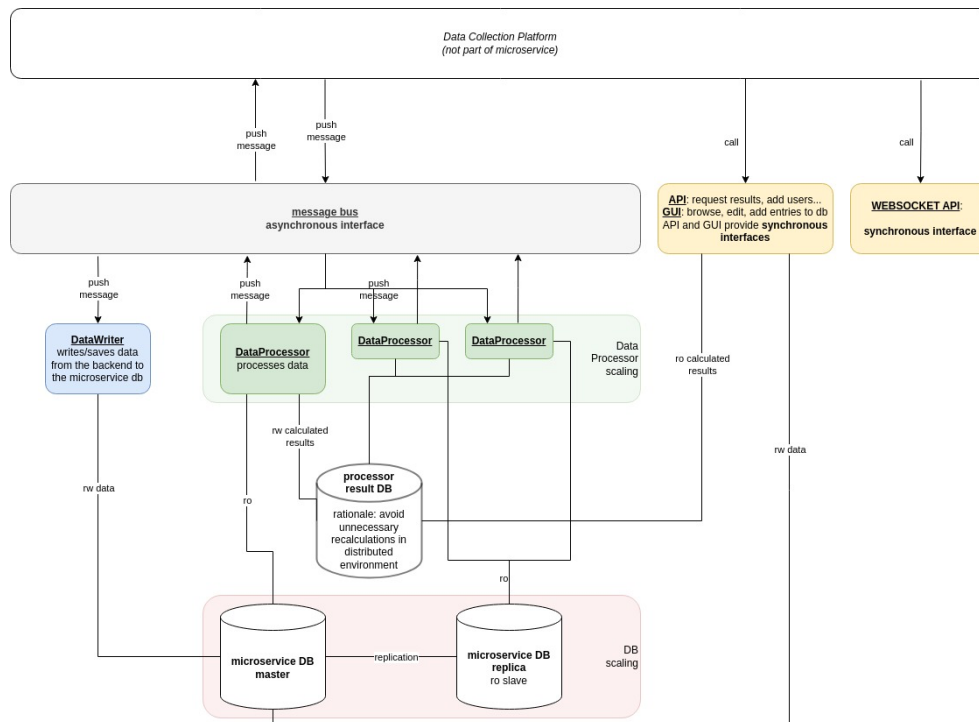


It is important to mention that:

- there is no direct link between the respondent and the microservices: the data collection platform has full control over microservice usage (who/when).
- there is no direct link between the microservices and the main database. This means that the data collection platform has full control over which data is delivered to the microservices. The exact mechanism which guarantees privacy will be explained in the “Regulation perspective” section.

Functional view

This diagram describes the different runtime functional elements of the microservice. Typically, each runtime element is deployed as a container (see Deployment View).



Main elements and their responsibilities are:

Message bus

The message bus allows for asynchronous communication between the data collection platform and the microservice.

Rationale:

- avoid blocking calls e.g. the platform must be able to quickly forward data (scanned receipt information, geolocation point) to the microservice without being blocked for too long. The message bus is able to fulfil this requirement by putting the data in a queue without any processing. In addition, by putting the message bus on the same server, networking issues between platform and bus are being avoided.
- notification service e.g. the DataProcessor can send a message that (some) data is processed. The platform can then take appropriate action.

Chosen technology: RabbitMQ (<https://www.rabbitmq.com/>)

API and GUI

The API and GUI are the synchronous interfaces of the microservice.

The API is used by the platform to fetch microservice data, request processed data results etc.

The GUI is used by a researcher or operator to:

- browse and inspect the results of the DataProcessor in the processor result DB
- browse, inspect and edit the data of the microservice DB

- possibly other functions e.g. add a scanned receipt and test the outcome

Because microservices have different functionalities, the (optional) GUIs are microservice-dependent. The GUIs are typically built with web technology and preferably share the same web framework than the API part.

Preferably, the GUI is integrated in the platform UI/backoffice in order to get an integrated user experience. This also avoids possible data inconsistencies between microservice database and platform database (e.g. a researcher edits the microservice DB but this change is not propagated to the platform database).

It is possible to extend the API with a synchronous call to the DataProcessor's internal algorithm i.e. without doing a request to the DataProcessor container. This is a simpler but more limiting design:

- the number of parallel requests might be limited by the webserver,
- it is a synchronous interface which means the call might block for a while,
- in case of networking issues, there is no queuing of requests or messages (in contrast to the message bus).

The synchronous call might be more practical than the asynchronous one for debugging the algorithm e.g. because no message bus is needed.

Websocket API

The websocket provides a synchronous interface as well. It can be used though to call the DataProcessor's internal algorithm synchronously. See discussing above.

Added value: if deployed, it is an independent product which can be directly used via internet.

Nice to have as a test platform.

Library vs service. Proposal: take into account in architecture/design (but no development yet).

DataWriter

Receives push messages with data from the platform via the message bus. The DataWriter writes the data in the microservice DB. Data push messages are queued in the message bus until the DataWrites is able to accept them.

There is only one DataWriter process in order to guarantee that the received data is written to the database in the same order as the data was pushed by the platform. This avoids (subtle) race conditions in which a DataProcessor start processing data with missing in-between data (e.g. a tracking point is missing in the DB between the first and the last tracking point).

Because the DataWriter only writes data to the database and doesn't process data (no CPU time), it is expected that it will be fast enough to always empty the queue. If this is not the case, platform design (and *not* microservice design) must be reconsidered e.g. by limiting the number of messages sent to the message bus.

DataProcessor

Does the real work of processing the data.

Starts processing when it receives a push message of what to process. Note that a single push message is delivered to one and only one DataProcessor. Scalability is achieved by load balancing the requests over the DataProcessors, which is a feature of the message bus. See the perspective on performance and scalability.

Given the required processing time needed by the processors, the platform should limit the number of messages sent to the data processors (via the message bus). In this regard, system design is important. E.g. in case of geo tracking, the processors shouldn't be triggered for each tracking point to recalculate the respondent's itinerary, rather, the tracking points should be bundled before the recalculation is done.

Because data processors act independently, the concurrency aspect of the data processor must be taken into account in its design. At an infrastructure level, limiting resources (e.g. in a k8s cluster) might be necessary.

A DataProcessor pushes a message on the bus when processing is done.

Processor Result DB

This DB stores the results of the DataProcessors. Since results can be re-calculated, persistent storage is not a strict requirement e.g. one might opt to make it a RAM only DB.

A non-sql database is probably most convenient to store the results.

It is accessed by the API/GUI element to fetch the results.

It can be consulted by DataProcessors to avoid the re-calculating of data, it therefore also act as a cache.

Chosen technology: REDIS (<https://redis.io/>). It is also possible to replicate a REDIS DB over several nodes if needed so (see perspective on performance and scalability).

Microservice DB

Used by the DataWriter to store pushed data.

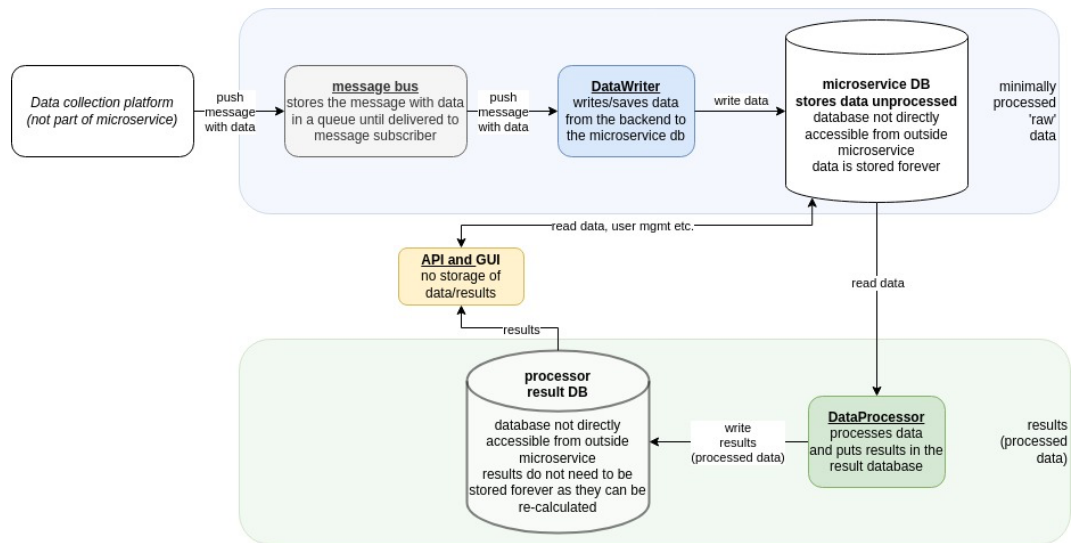
Data is consulted by the DataProcessors.

GUI is able to change data if needed so.

DB scaling is achieved by replication, see perspective on performance and scalability perspective.

Information view

This view describes the way that the microservice stores, manipulates, manages, and distributes information. The diagram highlights the key points.

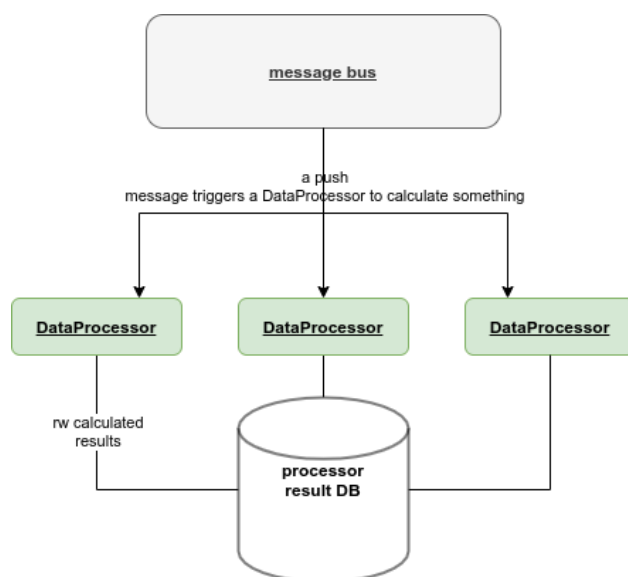


Concurrency view

This view identifies the parts of the microservice that can execute concurrently and how this is coordinated and controlled.

All functional runtime elements are allowed to run in parallel since their responsibilities are clear and non-conflicting (e.g. the DataWriter writes data while the DataProcessor processes data).

The most important concurrency aspect in the microservice architecture are the different DataProcessors which can process data in parallel.



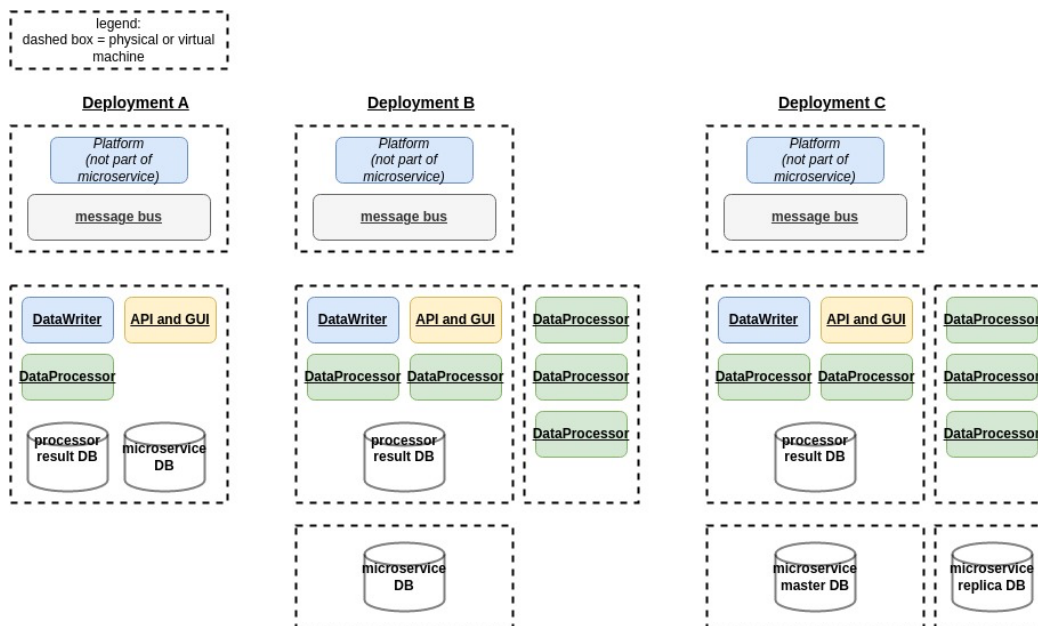
Care must be taken to:

- avoid race conditions: if 2 DataProcessor calculate the same thing, then one DataProcessor might overwrite the results of the other, also if the other's results were more recent.
- avoid unnecessary recalculations: DataProcessors should check the database to make sure the results for its calculation are not already there. In that sense, the processor result DB also acts as a kind of cache. If a single result is a combination of multiple small results, then cache optimizations might be possible by re-using the finer-grained results. E.g. suppose you need to calculate a timelog of a day. If a day is in progress, then possibly only recalculating the last hours is enough instead of recalculating the whole day.

Deployment view

A microservice is deployed as a collection of Docker containers: each functional runtime element is built as a Docker container. This makes all elements (almost) independent from the host OS.

Depending on the performance and scalability requirements (see perspective), different deployment strategies are possible. Here are some examples:



Note that the message bus should always be ready to accept messages from the platform (to avoid the platform to be blocked). To exclude networking issues, platform and message bus are on the same machine.

Operational view

Installation and upgrade

The microservice is a collection of Docker containers that will be managed, scaled and deployed with a container-runtime platform (e.g. Kubernetes <https://kubernetes.io/> for production environments, docker-compose for development etc.).

Backup and recovery

Two databases are (potentially) part of a microservice:

- processor result database: because results can be recalculated, no backup is needed here except to speed up the recovery process. If Redis is used as technology, then persistency is build-in. Backup/restore is the responsibility of the platform owner (and not of the microservice).
- microservice database: backup/restore is the responsibility of the platform owner.

Note that the choice can be made to store data sent to or received from the microservice *also* in the platform core database. When the microservice is disabled or not needed anymore, then the platform core can function without the microservice (e.g. the respondent's geo itinerary can be retrieved without the microservice).

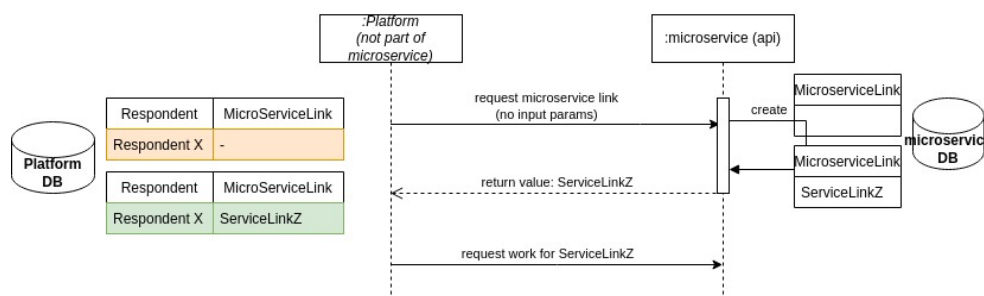
Perspectives

Regulation perspective

Privacy

Sensitive information must be restricted to the database of the main application. The microservice is not allowed to pull user/respondent private information into its own databases.

The following mechanism is foreseen:



The Respondent X entry is never visible in the microservice. Platform and microservice are linked to each other via a “microservice link”. The microservice only has knowledge of the abstract “microservice link”, which essentially is only an id (UUID, GUID...).

If the microservice database would be shared for researchers (e.g. for postprocessing), then the sensitive information of the respondent cannot be leaked.

Performance and scalability perspective

Depending on the application (e.g. type of research) and the type of microservice (e.g. processing intensive vs IO intensive), performance and scaling of the microservice can be tuned as follows:

- DataProcessors can be scaled:
 - by creating multiple instances
 - by distributing instances over multiple machines
 - see ‘concurrency view’ of how they distribute work

- microservice DB can be scaled:
 - by DB replication
 - by distributing the replication databases over multiple machines
- microservice results DB could be scaled similar to the main microservice DB. Since this is probably not a heavily loaded database (no complicated queries, only results), scaling might not be needed

4. Functional and non-functional requirements of the Receipt Scanning Microservice

This fourth chapter addresses the functional and non-functional requirements of the Receipt Scanning Microservice. The goal of the SSI project is to involve and engage households and citizens, and to define and operationalize a new/modified end-to-end data collection process.

Central to the SSI project stands the use of smart devices and other connected devices to obtain the data. NSIs and linked organizations have worked on platforms to allow households to register their purchases online. These platforms are @HBS by CBS, MOTUS by hbits as well as the developments at SSB and Insee.

An important criterion within the SSI project is the realization of an end-to-end data collection process, that results in qualitative and comparable data. The definition of quality and comparability stems from the mission of the ESS and trust upon the Principles of the European Statistics Code of Practice, which in its latest update also takes into account the emergence of new data sources and use of new technologies.

Within SSI, WP3 is the gateway to include smart data. The inclusion of smart data is seen as a need to further support the participation of the respondent in studies like TUS and HBS. In WP3 the Smart inclusion is realized by the development of microservices.

This chapter has a focus on Household Budget Survey (HBS) and the definition of requirements for the Receipt Scanning Microservice. The microservice is seen as a middle part software that is supportive to the household in reducing their burden to complete a consumption diary.

The microservice comprises two underlying microservices. The first (OCR microservice) is designed to perform Optical Character Recognition (OCR) and Document Understanding. The second is tasked with COICOP classification (COICOP classification microservice), with the objective of classifying a product/service to a COICOP category.

Note: when we refer to 'ticket' we also mean receipt or invoice, note, e-tickets, etc. All of these artefacts can be processed as an image, as long as the procedures were nationally trained for all variations.

Business requirements

HBS collects in a large detail what households spend on goods and services. In this way, the survey gives a picture of the living conditions and spending habits in the EU. HBS is performed by each Member State to calculate weighted macroeconomic indicators used for national accounts and consumer price indices. Eurostat publishes output since 1988 in intervals of 5 years. The last waves are from 2010 and 2015. In 2026 HBS will enter the IESS agreement and HBS will become a mandatory data delivery.

In the majority of member states, in a HBS study, (a member of) a household records tickets in a diary. Besides information on the store itself (name, address, logo, registration number, ...) a ticket at minimum holds information on the different purchases (or product rows; consisting of product name and product price) that are bought and the total price of the ticket. Depending on the shop

and product type, a ticket can also contain various different contexts to the purchase as well as information on reductions, amounts, units, return items or even empty good claims. The design of the diary defines the amount of detail that needs to be transferred to the diary.

This altogether creates a demanding effort from the participants to the study. Given the declining trend in participation rates and supported by the request of the Wiesbaden Memorandum, in 2011 Eurostat and the NSIs started to develop and implement new data collection modes to call a hold to this downward trend, and to even improve the quality of the collected data.

Initiatives of various countries, and previous EU-funded projects have translated the paper-and-pencil method to an online data collection process, giving households the opportunity to digitally respond to all questionnaires as well as digitalize their ticket by adding purchase by purchase in a step-by-step manner in order to submit the entire ticket into a digital diary.

Notwithstanding the added value of these online applications, the burden on the participants remains high, and the process is still too much error prone. The goal of WP3 is to reduce these gaps by developing and implementing microservices that acquire, process and (can) combine data collected from smart devices and other applications, in the case of HBS through the development of a receipt scanning microservice. This will transform the way digital diaries have been used so far and is aimed to result in a true added value of digitalization. Also for the (end) users.

A successful realization of the development and implementation will not entirely reduce the active participation of households in the registration of their tickets and purchases, but will provide support and guidance in their task to arrive to qualitative and comparable data for the ESS. It means that besides the development of the microservice also the implementation of the service to the platforms is important, as well as the UI/UX that presents the output of the microservice to the user, and the ease in which the user can verify, adapt, or even delete the output.

The following objectives are essential in reaching this goal:

- Objective 1: To define an architecture of a microservice (that is also to be reused in the other developments of WP3, being the geolocation and energy use microservices)
- Objective 2: To develop a receipt scanning microservice using OCR
- Objective 3: To implement classification solutions (string matching, machine learning, or search algorithm based) to classify purchases to a COICOP-list
- Objective 4: To develop an API to connect to/from other environments
- Objective 5: To deploy the microservice as a containerized application in the cloud
- Objective 6: To implement/integrate specific microservice parts in the app (e.g. algorithm). This integration should be feasible, should have an added value for the platform and/or should improve the user experience.

The stakeholders are the NSIs and their product owners (who represent the households (citizens)).

HBS study

In this section HBS studies are being described as they provide the context in which the Receipt Scanning Microservice operates.

In HBS studies questionnaires and a consumption diary are completed by the households. At the moment household members arrive in the diary phase they, at the least, already have completed a

questionnaire. If this member is the reference person, or the head of the household also a household questionnaire and a matrix to compose the household is part of the pre-diary tasks. All tasks are defined in a respondent journey or study flow that shows a sequence of tasks. Since the HBS diary setup requires an equal distribution of participation over the entire fieldwork period, and household members are requested to keep their diaries for the same period this study flow can be quite complex. This is without saying that different NSIs (can) make use of different data collection strategies, and that e.g. the questionnaire and the diary can be in a different sequence or taken with a different tool.

Central to a HBS study is the registration of tickets and purchases of goods and services in a diary. Households keep one diary over a period of (minimum) 2 weeks/15 days. Left aside paper-and-pencil diaries, a household member partakes in a HBS study via an application, be it via a mobile application, be it via a web application running in a browser.

HBS diary

The diary collects at the minimum:

- a description of the products and services that are bought (some countries also include the fixed (repeating) costs into the diary, others collect this information via a questionnaire)
- the price of each product or service, and
- the date of the purchases and periodicity of fixed costs

The registration of the products and services is linked to a COICOP-classification. COICOP stands for Classification Of Individual Consumption by Purpose.

The matching COICOP code is selected/mapped from a list:

- a COICOP-code²

In addition, on the level of the ticket extra information is/can be gathered:

- the country of purchase
- the shop (brand/type)
- ticket reduction
- professional purchase
- payment method

As an extra, on the level of the product or service extra information is/can be gathered (depending on national needs):

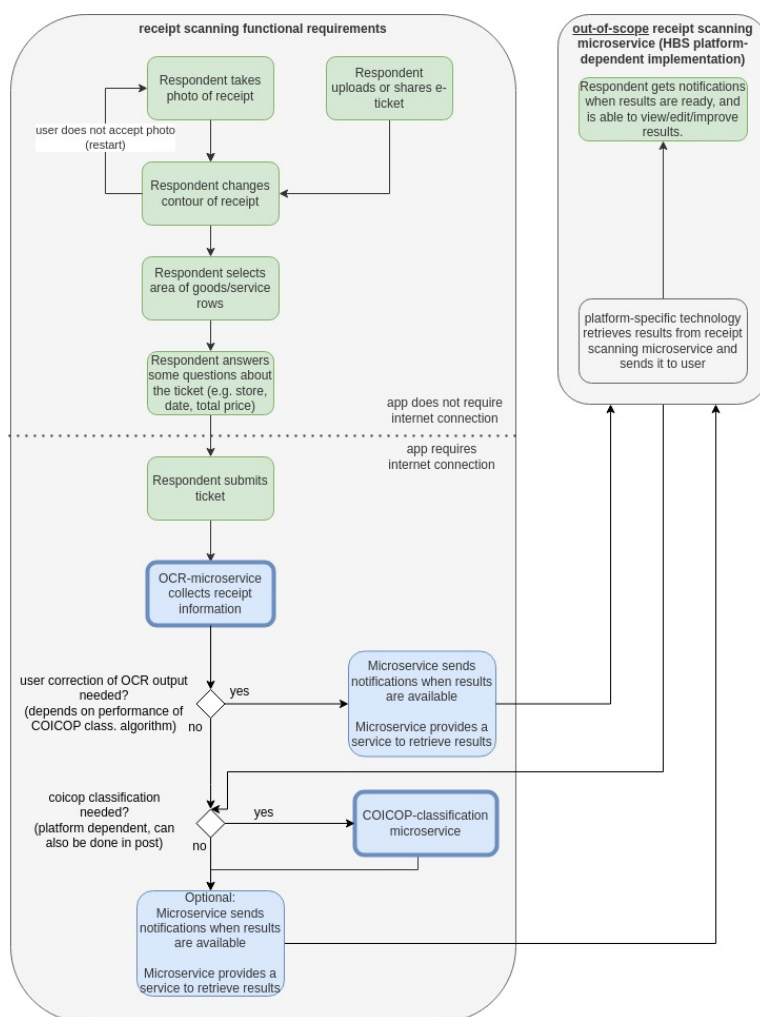
- number of items
- price per item
- quantity and metric/unit per item
- discount
- return

² Note: NSI's can decide to classify a product/service to a COICOP also after the data collection stage.

Functional requirements

The diagram gives an overview of the main functional requirements:

- functionality related to *user handling* is indicated by the green boxes. The respondent must be able to submit a photo or pdf and indicate to the receipt scanner software the receipt location in the photo/pdf and provide some receipt details for verification.
- functionality related to the *microservice* is indicated by the blue boxes. The essential function is to find and provide information that the HBS diary collects in a receipt (i.e. all bullet items in section 'HBS Diary').



User handling

Respondent handling (green boxes)	
REQ R1a	Respondent takes photo a receipt
	Select 'take a picture' to open the picture functionality
	Real-time camera opens:

	<ul style="list-style-type: none"> • detection of contrast ticket vs background • detection of light (good exposure) • detection of contour of ticket starts (4 dots or polygon around receipt when stabilized)
	App (or user) takes picture when stabilized (contour good enough)
	If the respondent is not satisfied with the photo, he/she needs to be able to restart with the photo taking process.
	Note that additional quality checks might (and probably will) be performed later by the app software itself and/or the server-part microservice.
REQ R1b	Respondent uploads an e-ticket with the application (alternative to taking a photo of a receipt)
	Select 'Submit e-ticket' to select/download a file from the local filesystem
	It is unclear yet which types of e-tickets will be supported. This strongly depends on the layout and structure of the ticket itself. Possible e-ticket formats to be supported: image, pdf+text(+variants) or pdf+image
REQ R1c	Respondent shares an e-ticket from another application (e.g. store app) to the HBS application (alternative to taking a photo of a receipt)
	Not part of microservice. Platform-specific (app) implementation.
REQ R1d	In case of a web app: scanning can be done by the browser of the smartphone and sent over to the browser running on a computer or laptop
	<p>Respondent is on the web app</p> <p>Respondent wants to take a photo <i>with the smartphone</i> so that it is automatically loaded in the <i>web</i> app</p> <p>This requirement is a nice to have.</p>
REQ R2	Respondent changes contour of receipt
	Respondent can change the contour (4 dots connected with lines) to define the ticket by moving dots or the line segment (handles) between two dots (parallel movement of two dots).
	Could be skipped if the automatic contour detection works very well.

	Ideally, this step is not necessary (same as R3).
	Having the complete receipt is important because it contains more info than only the product/service rows. Extra info on the receipt includes: store logo, store details, payment info etc.
REQ R3	Respondent selects other details of the receipt
	Selection of product/service rows. Helps the OCR process.
	Could be skipped if automatic product/service row detection works very well.
	Although extra work for the respondent, this step ensures the software knows the most relevant part of the ticket i.e. the product/service rows. Also, the positional data could be used later for ML training. Ideally, this step is not necessary (same as R2).
	This step does not involve a crop of the image, so at submission, the whole image will be sent to the microservice.
REQ R4	Respondent answers some questions about the ticket (questionnaire)
	The following questions will be asked: <ul style="list-style-type: none"> • Country • Shop • Language • Date • Total price
	Depending on the specific-platform UI, it must be possible to skip this step. Note however that the output of this step is very interesting for internal quality checks in the OCR process. Furthermore, knowing the store might/will be important for the COICOP classification.
REQ R5	Respondent submits ticket image
	A button allows the respondent to submit the ticket image, the change the contour hint and other selected areas (such as expense items).
	Different UI implementations are possible: <ul style="list-style-type: none"> • the image is uploaded in the background and the respondent gets a notification when it is done, or

	<ul style="list-style-type: none"> • a dialog should run to show the continuation of the upload. • Or, user settings whether he/she wants to send real-time or in background; or, via mobile or only wifi
	<p>Communication of success, or failure which has to be accepted by the user by pressing OK.</p> <p>In case of failure, the action that needs to be undertaken is UI/platform-dependent. E.g. one could wait for a wifi connection before trying to upload the image again. The decision what needs to be done is platform-specific.</p>

Temporally note: the above diagram was the initial functional view. The current state of the Receipt Scanning Microservice lowers the number of respondent tasks in comparison to what is listed in the diagram. For example, the user/respondent does not have to mark the contour of the ticket as the current version is very reliable in finding the correct contour and orientation via the first AI/ML model. Furthermore and in line with the first developments and tests, a well-trained document understanding model should be able to achieve a high hit rate of detecting the store, date and total price on the ticket, as well as the product rows.

It is important to mention though that an AI model never provides guarantees. If certain input (e.g. shop name) is *required or essential*, then a respondent's input or correction is necessary.

Microservice

Microservice (blue boxes)	
REQ M1	Microservice collects receipt information
	<p>The service is best effort and should try to collect the following receipt information:</p> <ul style="list-style-type: none"> • date of the purchases • a description and price of the products and services that are bought • the country of purchase • the shop (brand/type) • ticket reductions • payment method <p>Then, at the level of a product or service:</p> <ul style="list-style-type: none"> • number of items • price per item • amount and metric/unit per item e.g. 1,5 L • discount • return or not
	Information retrieved from the user in step (e.g. R2, R3 and R4) could be used as a

	verification step. E.g. the total price as answered by the respondent should match the total price as derived from the image. If not, the user's input has priority (esp. in the UI as we don't want to overrule user's input).
	Depending on the performance and quality of M1, the number of respondent actions in the UI (more specifically, R2, R3 and R4) might change (e.g. if the service is almost always able to retrieve the product/service rows then step R3 is probably not needed anymore).
	Because the output (receipt information) might contain several text mistakes, it might be desirable to let the user correct those mistakes before the receipt information is handed over to the COICOP classification algorithm. Whether or not user correction is desirable strongly depends on the input requirements of the COICOP classification algorithm.
REQ M2	Microservice performs COICOP classification on detected products and services
	The model (algorithm + training) classifies the product/service description to a COICOP.
	Integration of the COICOP Microservice together with the OCR Microservice.
	Support for different COICOP classifications (the code should not hard-code one specific COICOP classification since NSIs are free to extend the 5-digit demanded classification).
REQ M3	Microservice sends notifications when results are available
	So that a pop-up in the app can inform the respondent that the scanned ticket is added to the overview on the day of the purchases.
	In case of multiple submitted receipts, the microservice will generate a notification for each receipt. It is up to the app (platform-specific) how to handle multiple notifications.
REQ M4	Microservice provides a service to retrieve results
	The output of the microservice can be requested by platform to e.g. include the user into quality control.

Non-functional requirements

Non-functional requirements	
REQ N1	The microservice should be independent from any specific HBS platform.

	The microservice has no dependency to other environments, and has an independent operation.
REQ N2	It must be possible to connect and communicate with the microservice from any HBS platform.
	The microservice receives input, and provides output making use of APIs.
REQ N3	The microservice must have a design in which algorithms (computer vision, AI, ML) can be easily improved/updated.
REQ N4	The service must be deployable at any institute/NSI (shareability).
	The microservices are provided as software packages in containers, which can be easily shared and deployed. Docker is a software that can host containers. Kubernetes is often used as software to orchestrate various containers.
REQ N5	The service must be scalable with the number of receipts it needs to handle.
	Kubernetes is a software used to orchestrate containers. By this Kubernetes allows to horizontally scale the containerised microservice depending to the number of receipts received.
REQ N6	Security by design
	Using the container technology barriers are created between various components used in the study setup, which deliver better privacy, security and maintainability, scalability and high availability. Communication between the platforms runs through APIs and https communication.
REQ N7	Privacy by design
	Using the container technology barriers are created between various components used in the study setup, which deliver better privacy, security and maintainability, scalability and high availability. Communication between the platforms runs through APIs and via UUIDs to avoid transferring personal information.
REQ N8	Support for localization
	Algorithms being applied by the microservice should be configurable or trainable (in case of ML) to support localization, which includes different languages, different currencies, date formats, dots vs commas etc. This is required to make the

	microservice shareable.
REQ N9	Offline vs online support (app)
	Parts of the microservice are/can be selected to be developed in a Library to run offline in an application. The library must take into account platform-dependency (Angular, ionic, Flutter ...) to function.

5. OCR microservice (Receipt Scanning Microservice – part 1)

The Receipt Scanning Microservice is supported by two microservices: the OCR Microservice (part 1) and the COICOP Microservice (part 2).

Chapter 5 deals with the OCR Microservice holding three AI/ML models. This part is non-domain specific. In this chapter the software design, software implementation, and platform integration of the OCR Microservice are discussed. Software design and implementation explain the inner workings of the microservice, while platform integration explains how the microservice should be integrated technically in a platform. The responsibility for platform adaptations and UI elements lies with the platform developers and is not part of the SSI scope. Currently, the integration of the OCR microservice by hbits (MOTUS) and CBS (@CBS) is taking shape. INSEE and SSB are asked to look into the documentation and will report on the feasibility to integrate the microservice into their production environment. Other countries are invited to request information on how to integrate this shareable component into their platforms.

Code is to be found in the Git repository under the file [TODO next DL].

Demonstrations that were given are available on OpenSocial via <https://cros.ec.europa.eu/book-page/information-session-march-2024-wp3>.

As a last point this chapter will discuss test information.

Design

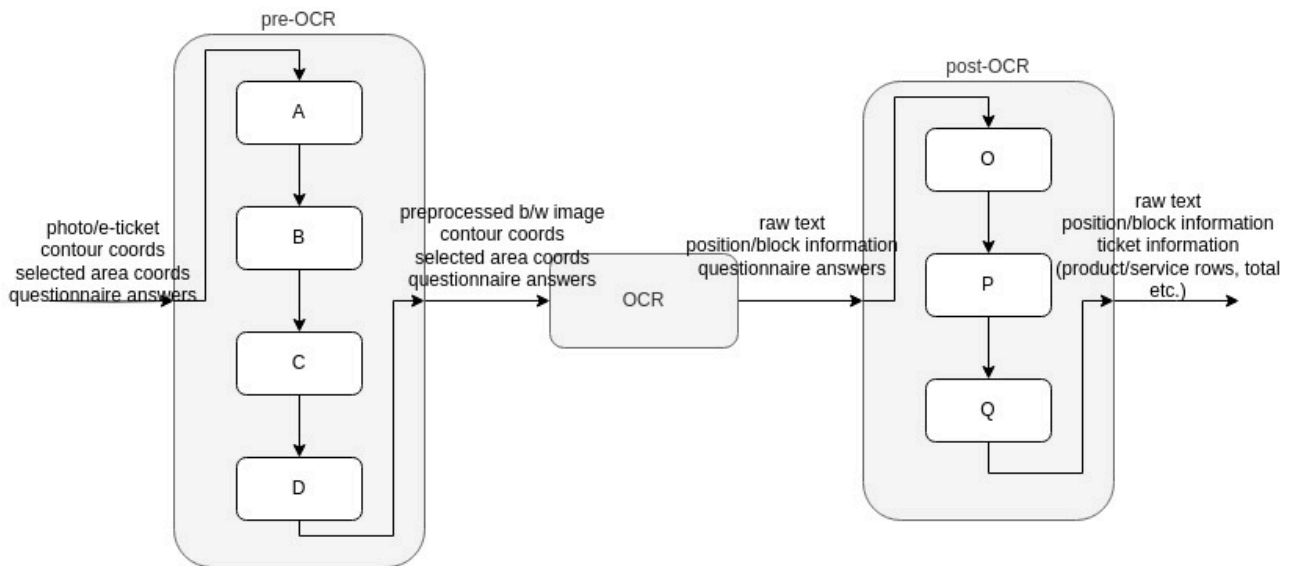
The core of the microservice is the OCR pipeline which takes a receipt in the form of:

- a set of images (typically only one image), or
- a (pdf) e-ticket (which accordingly is processed as an image)

Once the receipt has arrived it goes automatically and sequentially through 3 steps:

- Pre-OCR: pre-processing the received image or pdf
- OCR: optical character recognition
- Post-OCR
 - document (receipt) understanding, and
 - final post-processing which has as an output a json file.

The OCR Microservice returns all available information, so that that information can be used by the platform (and its UI/mobile and web application) and or/accordingly to the COICOP Microservice. For example, if a product row cannot be parsed (e.g. price cannot be found), it returns the whole (unparsed) row.



The letters (A-D) indicate a pre-OCR processing function/block in the whole chain. E.g. a noise removal function, an receipt detection function, etc. The letters O-Q are similar but then for post-OCR e.g. document understanding, combining detected rows, validating the total price etc.

Pre-OCR

The main responsibilities of pre-OCR step are receipt detection, orientation correction and image cropping. Other techniques like noise removal were listed in a MOSCOW analysis and were taken in consideration but did not improve the results and are therefore not taken on board.

The input is a single receipt in the form of:

- a set of images (this means that the microservice can deal with multiple images), or
- an e-ticket (pdf). The pdf is converted into a set of images which then follows the same flow as normal images.
- Optionally: contour coordinates

The output of the pre-OCR step is a correctly oriented, cropped receipt image.

For receipt detection³, two methods have been deployed:

- semantic segmentation, and
- object detection

³ For these processes use have been made of open-source code and models. This is also true for PaddleOCR. PaddleOCR stands as an open-source optical character recognition (OCR) solution crafted by PaddlePaddle, the deep learning platform nurtured by Baidu. Its primary objective is precise text extraction from images, boasting proficiency across diverse languages and font types. Employing cutting-edge deep learning architectures, PaddleOCR excels in both text detection and recognition tasks. Its adaptability shines through the provision of several pre-trained models, each tailored for distinct scenarios like scene text, ID card, and table structure recognition. Offering standalone models and end-to-end OCR pipelines, it accommodates various use cases and deployment settings. Appreciated for its user-friendly interface, robust performance, and comprehensive documentation, PaddleOCR has garnered favor within research and developer circles. Its utility spans document digitization, image text extraction, and intelligent document processing. Furthermore, its open-source nature fosters community engagement, permitting customization to suit specific language or application needs.

Semanantic segmentation

TODO next DL

1. The pre-ocr pipeline takes as input an image of a receipt.
2. It then segments the image. Which means that it labels each pixel of the receipt as belonging to the receipt or not.
3. It then draws the smallest possible bounding rectangle around the pixels classified as belonging to the receipt.
4. It uses heuristics to try and correctly orient the receipt, such that it is upright.
5. It crops the receipt using the bounding rectangle.
6. It then adjusts the orientation again slightly, using `paddle_ocr` to fine tune the orientation.

Model training

TODO next DL

Object detection

An alternative method is to use object detection for receipt detection. In this case a rectangular box is drawn around the receipt in the image. The detection is very accurate but less fine-grained than semantic segmentation. Furthermore, it requires quite some processing to correct the receipt orientation. The orientation of the image is corrected by sequentially applying PaddleOCR to derive text orientation.

The used AI model for object detection is YOLOS. For more information please visit this link:

https://huggingface.co/docs/transformers/model_doc/yolos

Model training

The training of the object detection model holds 4 steps:

- Collect images and divide them into 3 subsets: train, test and validation.
 - Train: for training the YOLOS model,
 - Test: for testing the YOLOS model while it is being trained,
 - Validation: for validating the final YOLOS output (which is not used by YOLOS itself).
- Rotate all images: This step is needed to make the crop of the receipt as tight as possible.
- Annotate and create Coco output: Use a tool to add bounding boxes for object detection to the images (e.g. Label Studio), export in Coco format (note: validation images should not be annotated because they are not being used in YOLOS training).
- Train and retrain

More detailed description can be found in the Git repo: [TODO next DL].

OCR

TODO next DL

Model training

TODO next DL

Post-OCR

The output of the OCR step includes text and text locations (bounding boxes). That information is used:

- to understand the receipt i.e. trying to give a meaning to the recognized text (by OCR),
- to correct OCR mistakes (TODO next DL - date, time corrections etc.),
- to produce a final json output which contains all receipt details (and some metadata).

Receipt understanding

In order to understand the receipts, the pipeline applies a fine-tuned model of LiLT. See also here for more information: <https://arxiv.org/abs/2202.13669>.

LiLT combines text and layout (text position) information to label a text box. A variety of labels were defined within the SSI project to arrive to standardisation, ranging from store address to tax price:

Id	Label	Description
0	O	ignore
1	l-date_text	date string e.g. Date
2	l-date_value	date e.g. 10/9/23, Thursday 3 aug 2019
3	l-time_text	time string e.g. Time
4	l-time_value	time value e.g. 10:29
5	l-datetime	combination of date and time (because combined by OCR) e.g. 3-12-2021 15:04
6	l-heading	Main heading of the ticket, typically between store details and products e.g. Receipt, Account, rekening, klantenbon etc., to distinguish from item.header
7	l-unused8	
8	l-unused7	
9	l-unused6	
10	l-tax.header	tax table: header e.g. Tax, Incl., Excl.
11	l-tax.description	tax table: typically percentage or total e.g. 10%, total
12	l-tax.price	tax table: tax price e.g. 9 EUR (which is 10% of 90 EUR)
13	l-tax.price_excl	tax table: total price excluding tax i.e. total cost without tax e.g. 90 EUR
14	l-tax.price_incl	tax table: total price including tax i.e. what the customer had to pay e.g. 99 EUR
15	l-unused5	
16	l-unused4	

17	l-unused3	
18	l-unused2	
19	l-unused1	
20	l-store.name	name of the store e.g. lidl, Coopcentrum Bert Stuut
21	l-store.address	address e.g. 9693 AE Bad Nieuweschans, Hoofdstraat 41
22	l-store.phone	phone e.g. Tel:0597-621678
23	l-store.email	email e.g. Email:stk@jumbo.com
24	l-store.website	website e.g. WWW.KWANTUM.NL
25	l-store.tax_id	tax identification number e.g. B0840.591.904
26	l-store.unused3	
27	l-store.unused2	
28	l-store.unused1	
29	l-store.etc	belongs to store but combination of several items e.g. Rotterdam 010 414 46 98 (which is part of address and tel. nr)
30	l-item.header	product table: header e.g. EUR, TOT, Price, Description
31	l-item.quantity	product table: quantity, how many items of this product e.g. 2 OR wieght e.g. 0.213kg
32	l-item.description	product table: description e.g. tomatos
33	l-item.unit_price	product table: unit price e.g. 1.00, 1.02 EUR/kg
34	l-item.price	product table: total price of this item row e.g. 2.00 (which is 2x 1.00 in this example)
35	l-item.id	product table: number, code or id of item e.g. 2751338001839, Article 20470047
36	l-item.discount_description	product table: kind of discount e.g. set 2 for 9.99, DISCOUNT
37	l-item.discount_price	product table: discount price e.g. -1,31, 1.31
38	l-item.etc	product table: sometimes contains non-product items e.g. BONUS CARD, Parking ticket etc.
39	l-item.unused11	
40	l-item.unused10	reserved for e.g. multi-line support

41	l-item.unused9	
42	l-item.unused8	
43	l-item.unused7	
44	l-item.unused6	
45	l-item.unused5	
46	l-item.unused4	
47	l-item.unused3	
48	l-item.unused2	
49	l-item.unused1	
50	l-sub_total.text	subtotal string e.g. SUBTOTAL
51	l-sub_total.price	subtotal price e.g. 95
52	l-sub_total.discount_text	receipt discount string e.g. DISCOUNT, KORTING
53	l-sub_total.discount_price	receipt discount price e.g. -3,99
54	l-sub_total.discount_item_text	discount item table (overview of discounts, similar to product items but then for discounts): text e.g. discount
55	l-sub_total.discount_item_price	discount item table: price e.g. -45,6 EUR
56	l-sub_total.tax_text	tax text (not from the tax table! see line 10-14) e.g. TAX
57	l-sub_total.tax_price	tax price e.g. 9 EUR
58	l-sub_total.tax_excl_text	total price excluding tax (string) e.g. Ex TAX
59	l-sub_total.tax_excl_price	total price excluding tax (price) e.g. 90
60	l-sub_total.tax_incl_text	total price including tax (string) e.g. TOTAL incl TAX
61	l-sub_total.tax_incl_price	total price including tax (price) e.g. 99
62	l-sub_total.service_text	service text e.g. SERVICE 3%
63	l-sub_total.service_price	service price e.g. 3 EUR
64	l-sub_total.item_rows_text	total number of item rows (string) e.g. ? (remove?)
65	l-sub_total.item_rows_value	total number of item rows (value) e.g. 10 (remove?)

66	l-sub_total.quantity_text	total number of items (value) e.g. ITEM COUNT
67	l-sub_total.quantity_value	total number of items (value) e.g. 20
68	l-sub_total.etc_text	related to subtotal (string) e.g. ROUNDING
69	l-sub_total.etc_price	related to subtotal (price) e.g. 0,00
70	l-total.text	total string e.g. TOTAL, TOTAAL
71	l-total.price	total price (typically incl tax) e.g. 99,7
72	l-total.rounded_text	rounded total (in case cash cannot be paid in certain amounts) e.g. ROUNDED TOTAL
73	l-total.rounded_price	rounded total price e.g. 100
74	l-total.unused4	
75	l-total.unused3	
76	l-total.unused2	
77	l-total.unused1	
78	l-total.etc_text	related to total but no other correct label (remove?)
79	l-total.etc_price	related to total but no other correct label (remove?)
80	l-payment.cash_text	string which indicates payment in cash e.g. CASH
81	l-payment.cash_price	value of cash payment e.g. 100
82	l-payment.change_text	change string e.g. CHANGE
83	l-payment.change_price	amount of change e.g. 1.00
84	l-payment.other_text	other payment type e.g. CARD, MEASTRO
85	l-payment.other_price	other payment price e.g. 99
86	l-payment.details_total_text	payment details (esp credit card details on the receipt) can also contain the total price e.g. total
87	l-payment.details_total_price	payment details (esp credit card details on the receipt) can also contain the total price e.g. 100,00
88	l-payment.etc_text	related to payment but no other correct label (remove?)
89	l-payment.etc_price	related to payment but no other correct label (remove?)

Model training

The training of the receipt understanding model holds 4 steps:

- Collect receipts: Images need to be correctly oriented and cropped. Collect receipts in 3 sets:
 - train: for training the LiLT model,
 - test: for testing the LiLT model while it is being trained,
 - validation: for validation after the LiLT has been trained. The validation set does not need to be annotated (except if needed for automatic regression testing).
- Apply OCR to get boxes with text: Tools: e.g. PaddleOCR.
- Annotate and compose the dataset. Every OCR detected text box must get a label (see table of labels in previous section, and please mention the O label, first row). Tools: e.g. Excel.
- Train and retrain.

More detailed description can be found in the Git repository:

`ocr_microservice/src/ocr_microservice/model_training/lilt/README.md`.

Corrections

[TODO next DL]

Final post-processing

Given the output of receipt understanding and its corrections, a json output is being generated which contains all found information as well as metadata (i.e. where the information came from).

The json output has the following fields (for explanation please view the aforementioned label list):

- date
- time
- tax_table which has rows with following fields:
 - description
 - price:
 - price_excl
 - price_incl
- store which has next fields
 - name
 - address
 - phone
 - email
 - website
 - tax_id
 - etc
- item_table which has rows with following fields:
 - quantity
 - description
 - unit_price
 - price
 - id
 - discount_description
 - discount_price
 - etc

- sub_total
 - price
 - discount_price
 - discount_item_text
 - discount_item_price
 - tax_price
 - tax_excl_price
 - tax_incl_price
 - service_price
 - item_rows
 - quantity
 - etc_price
- total
 - price
 - rounded_price
 - etc_price
- payment
 - cash_price
 - change_price
 - other_price
 - details_total_price
 - etc_price

Simplified example of a json output:

```
{
  "receipt": {
    "date": {
      "text": "23-04-2024",
      "corrected": "2024-04-23",
      "bbox": [[422, 2253, 574, 2280],[422, 2253, 574, 2280]],
      "ocr_confidence": 0.9,
      "du_confidence": 1.0
    },
    ...
  }
}
```

Explanation of fields:

- text: text as recognized by the OCR engine
- corrected: post-processed text e.g. normalized date
- bbox: bounding box as detected by the OCR engine
- ocr_confidence: OCR engine confidence score

- `du_confidence`: document understanding confidence score

Implementation

The OCR pipeline is a python runtime. The code is available in a Git repository. The code also includes an example Docker file for integration into a platform as is discussed in the next section.

Integration

Depending on the platform, the python runtime can be deployed as a (containerized) microservice in various ways. In the following sections, the data collection platform builders discuss their specific integration. In sequential order these are the platform of hbits, CBS, Insee and SSB.

hbits MOTUS platform

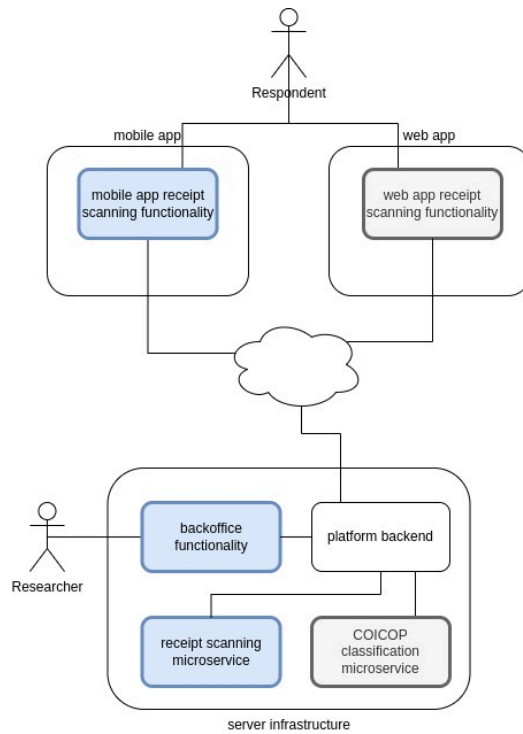
MOTUS is developed by hbits, as a spin-off of the Vrije Universiteit Brussel. How the integration of the OCR microservice is viewed on the user side can be seen in the demonstration videos (see xxx) provide a view on how the MOTUS application presents, user side, the output of the OCR microservice. Further information becomes available via the user tests in WP2.

Below the integration of the OCR microservice in MOTUS is discussed, as well as how MOTUS communicates via its API.

Integration in MOTUS

The integration in MOTUS discusses the views as explained in Chapter 3 of the Microservice software architecture.

Context view



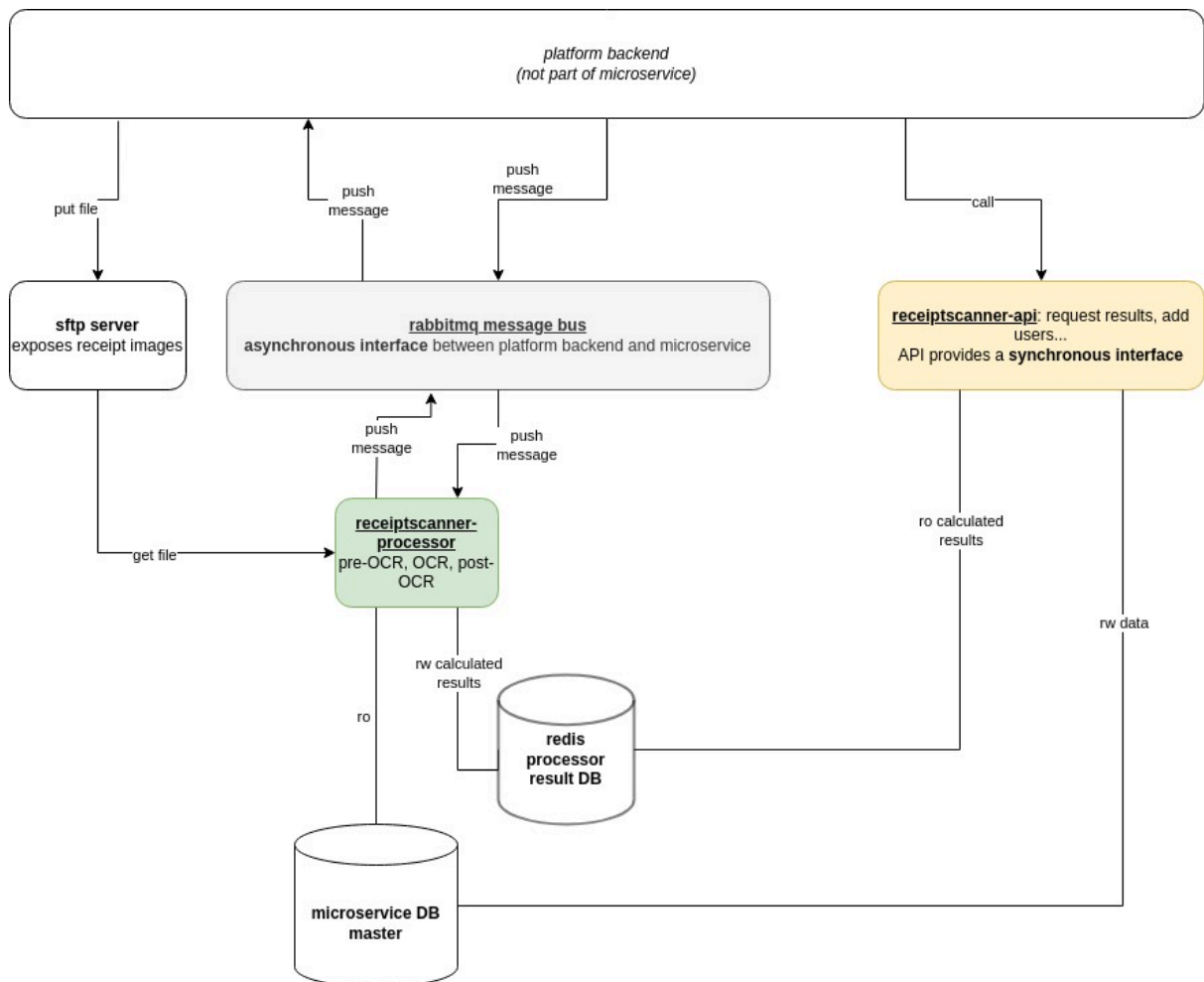
Functional view

Component	Functionalities
Mobile app receipt scanning functionality	<ul style="list-style-type: none"> • Camera view (take photo) • Gallery view (select image from gallery) • Pdf view (select pdf from local phone storage) • Photo view + contour editing • Pre-process image: decrease resolution • Send photo + coordinate data to backend
Web app receipt scanning functionality	<ul style="list-style-type: none"> • Scanning functionality is not (yet) foreseen. The output of the OCR microservice is nevertheless available via the web app.
Receipt Scanning Microservice	<ul style="list-style-type: none"> • From receipt photo + coordinate data to receipt information (store, total, product/service rows, etc.)

Component	Functionalities
Mobile app receipt scanning functionality	<ul style="list-style-type: none"> • Camera view (take photo) • Gallery view (select image from gallery) • Pdf view (select pdf from local phone storage) • Photo view + contour editing • Pre-process image: decrease resolution

	Send photo + coordinate data to backend
Web app receipt scanning functionality	Scanning functionality is not (yet) foreseen. The output of the OCR microservice is nevertheless available via the web app.
Receipt Scanning Microservice	From receipt photo + coordinate data to receipt information (store, total, product/service rows, etc.)
Backoffice functionality	Researcher can review respondent receipt
Respondent functionality	Respondent receive tentative data via the mobile and/or web app, can edit the ticket to commit the ticket

The diagram below shows the functional view of the receipt scanning microservice itself:

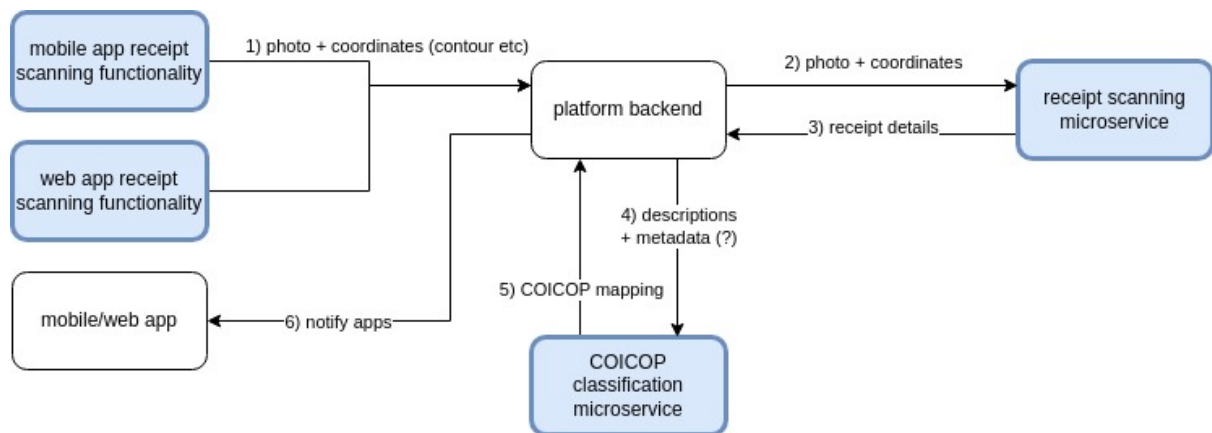


The components have the following functionalities:

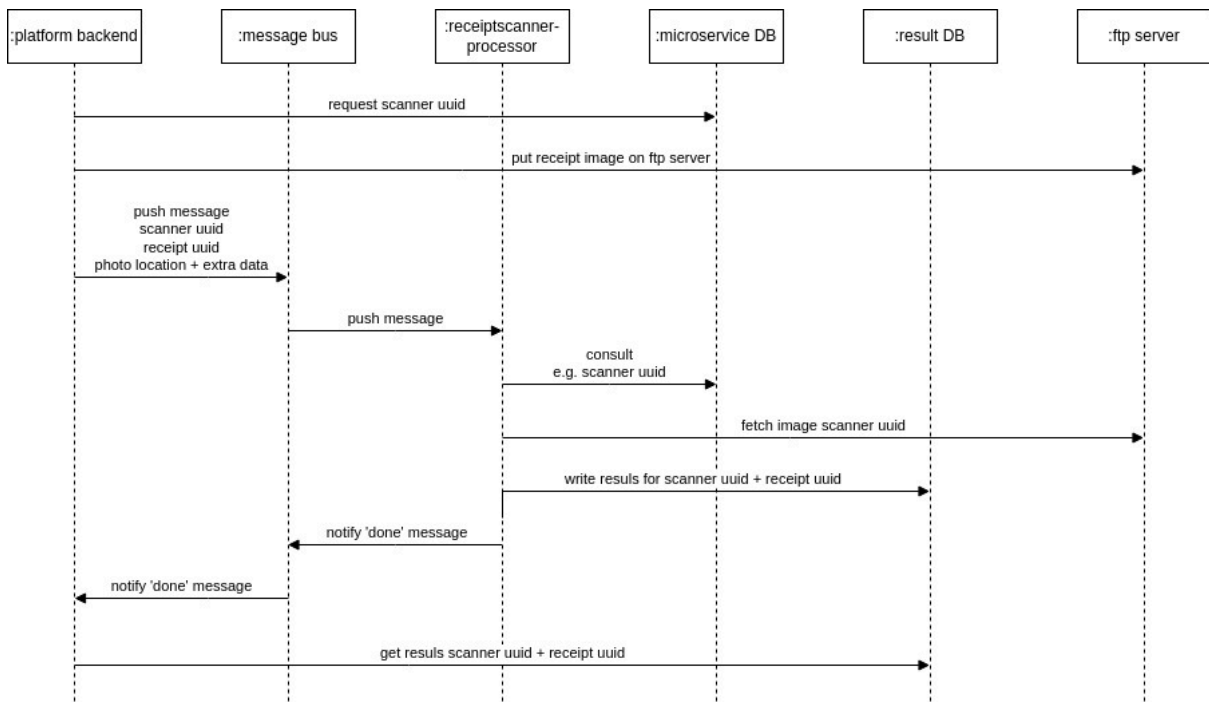
- *Sftp server*: makes receipt images available to the microservice

- *Rabbitmq message bus*: used for asynchronous communication between backend and microservice. Messages are requests for processing and notifications when processing is finished
- *Mariadb microservice database*: stores scanners (which can be coupled to respondents in the backend)
- *Redis result database*: stores processing results
- *Receiptscanner-processor*: downloads a receipt from the sftp server, processing it (pre-OCR, OCR and post-OCR), publishes the results in the redis database and notifies the backend via rabbitmq
- *Receiptscanner-api*: used by the backend to manage scanners (which can be coupled to respondents in the backend) and to retrieve results

Information view



The diagram below shows the informational view of the receipt scanner microservice itself:



Flow:

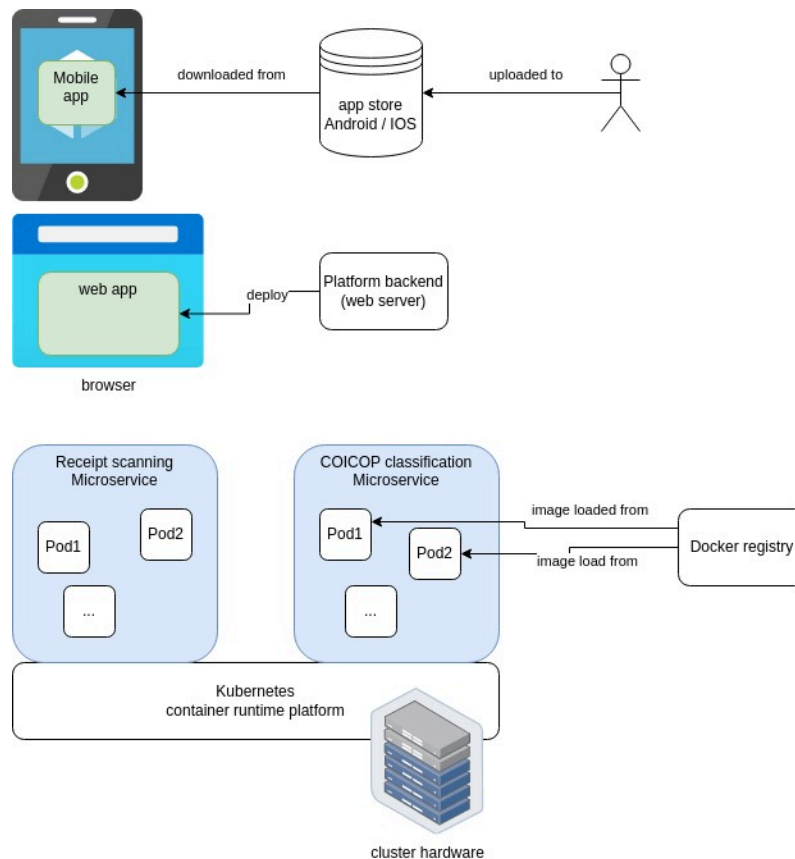
- Backend requests the receiptscanner-api to create a scanner entry for a respondent (if Receipt Scanning Microservice is allowed and activated for that respondent via the backoffice configuration)
- When the backend receives a receipt image
 - it saves the image on the sftp server. Files are organized by scanner-UUID directories. Filenames are date and UUID based.
 - it sends an asynchronous message (*receipt.process*) on the rabbitmq bus to process the receipt image.
- Receiptscanner-processor
 - reads the processing request message from rabbitmq
 - fetches the receipt image via sftp
 - processes the image
 - puts the results in the redis db
 - notifies the backend that processing is done via a rabbitmq asynchronous message (*receipt.processed*)
- Backend
 - reads the notification message
 - requests the results from the receiptscanner-api
 - returns the results to the respondent when requested by the mobile/web app.

[Concurrency view](#)

In mobile/web app is respondent identification via authentication used to isolate backend requests from each other.

The platform backend and microservices are able to handle multiple requests concurrently. The generic microservice architecture (see chapter 3) supports concurrent and parallel handling of requests by queuing requests and scaling data processors.

Deployment view



The components are

- Mobile app: via app stores
- Web app: loaded via browser from the web server which is part of the platform backend
- Microservices: as pointed out in the generic microservice architecture, a microservice is a collection of Docker containers. The receiptscanner containers are deployed on the MOTUS Kubernetes cluster platform (except for COICOP Microservice -- see later).

API

The asynchronous interface consists of 2 rabbitmq messages:

- The microservice listens for a ProcessReceipt message with format:
 - scanner_uuid: the scanner (~ mapping of respondent)
 - receipt_uuid: identification of the receipt
 - receipt_filename: name of the receipt on the ftp server
 - timestamp of format 'Y-m-d\TH:i:sP'
 - user_selection: optional array of x,y coordinates indicating the corner points which were assigned by the respondent

- When processing is finished the microservice pushes a ReceiptProcessed message on the bus:
 - scanner_uuid: the scanner
 - receipt_uuid: identification of the receipt

The MOTUS backend can then fetch the results via the synchronous REST interface, which has the following functions:

- GET scanners: returns the list of scanners
- POST scanners: create a new scanner
- DELETE scanners: delete a scanner-uuid
- GET scanners/{scanner_uuid}: show a specific scanner-uuid
- GET scanners/{scanner_uuid}/receipts: get receipts of a specific scanner-uuid
- GET scanners/{scanner_uuid}/receipts/{receipt_uuid}: get specific receipt of specific scanner
- DELETE scanners/{scanner_uuid}/receipts/{receipt_uuid}: delete specific receipt

CBS platform

[TODO next DL: integration & API]

Insee

[TODO next DL]

SSB

[TODO next DL]

Test

[TODO next DL]

6. COICOP classification microservice (Receipt Scanning Microservice – part 2)

The Receipt Scanning Microservice is supported by two microservices: the OCR Microservice (part 1) and the COICOP Microservice (part 2).

This section describes the COICOP Microservice, which has the task of assigning a 5-digit COICOP code to each product row that has been extracted from a receipt using the OCR Microservice. Hence, the COICOP Microservice is a classical text to classification task, integrated into the Receipt Scanning Microservice. The scope of the SSI project is to develop the technical possibility to integrate one or multiple different COICOP classification techniques into the Receipt Scanning Microservice.

This means that NSI-specific models will have to be developed by each NSI and training is not part of the SSI. Yet, different modelling techniques and methodological considerations will be outlined to inform data scientists and methodologists who want to deploy their own NSI specific model into the OCR Microservice.

Additionally, it is important to note that the use of the COICOP Microservice is methodologically recommended but technically not strictly necessary. The Receipt Scanning Microservice could also be used without the COICOP Microservice. This would then result in the need to manually or automatically (but independently from the Receipt Scanning Microservice in a separate system) classify each product row to COICOP in a later step. One reason could be that a given NSI might not have access to the required training data to develop their own COICOP classification model for receipt data. In essence, the Receipt Scanning Microservice would then turn into a mere data collection tool, still reducing the participation burden, but not using its full potential.

The following section will describe the current state of developments and conceptual ideas. Data from CBS and Destatis have been used to inform the use case. All considerations have been described in the most general way possible. Yet, one limitation is that receipt structures and data availabilities (e.g., CPI scanner data) vary fundamentally between various member countries.

Design

For the COICOP classification process, three distinct techniques are available: (1) automatic string matching, (2) machine learning, and (3) manual string searching. These techniques can utilize different training/source materials, which include: (A) gathered receipt data, (B) price statistics scanner data, and (C) manually curated tag lists. For some stores, there might be overlap between sources A and B where scanner data contain text information that is equivalent to the receipt texts.

Automatic string matching, machine learning, and manual string searching can all be used to link a receipt text row to a COICOP code. Automatic string matching (1) involves directly comparing receipt text to given source materials, offering simplicity and efficiency. This technique is straightforward to implement; however, it struggles with variations in spelling, abbreviations, and typos. Machine learning (2), on the other hand, uses models trained on the source materials, i.e., the receipt texts and the corresponding COICOP labels. As a result, the model learns to recognize more generic patterns and structures with which it can classify previously unseen receipt texts to a certain degree. The downside is the significant initial effort required for setting up a training dataset and a training pipeline as well as the need for substantial computational resources, especially if there is a need to

train multiple models (see below). Finally, manual string searching (3) involves the respondent in the coding process. In this solution, each product row of the receipt is entered into a search algorithm and the respondent selects the best-fitting COICOP category. This method is the most burdensome for the participant but can be highly precise since it leverages the respondent's expertise on their own private purchases. Also, the respondent could change the search string if the receipt text itself is not diagnostic enough. All three techniques heavily rely on the comprehensiveness of training/source materials and require continuous updates and refinements as new data becomes available, thereby adapting to changes in product listings.

The training/source materials (training data) that can be used with all three techniques can be gathered from three different sources. First, gathered receipt data consists of transaction records collected directly from consumers or retailers. A crucial step in this process is the hand coding of all extracted products to COICOP categories prior to using the data with any of the three techniques. This data source can be representative of the most consumed goods, but it is nearly impossible to capture all products this way. Second, price statistics scanner data, on the other hand, is collected from retail scanners at points of sale, typically used for compiling national price indices. This data is comprehensive and systematically collected but often limited to specific product categories like food and near-food items, which can constrain its applicability across various retail sectors. In most NSIs, automated COICOP classification systems are already in place, resulting in pre-coded training data. Last, manually curated tag lists are databases of product names and descriptions created and maintained by experts within the NSI. These lists require continuous updates and may not cover the full range of products found in dynamic and varied receipt data. Each of these data sources offers unique advantages and challenges, influencing their effectiveness in COICOP classification processes.

Regardless of the actual technique used, the training data must be similar or equal to the receipt texts extracted from the ticket. It is also essential to ensure that the training data covers product information on the store types that are allowed to be used with the microservice. For instance, if training data is only available for food and near-food products, receipts from hardware and home improvement stores should not be forwarded to the COICOP step 2 microservice. Ultimately, one maximization strategy could be to combine as many data sources as are available in a given NSI. Yet, whether combining data sources is effective should be carefully evaluated.

If a given NSI has the training data and technical resources, WP3 recommends using all three techniques in sequence. The process can be divided into five sub-steps within the COICOP classification microservice, where each subsequent sub-step is initiated only if the current one does not yield a satisfactory result for a given product row. Note that the criteria for such thresholds are country-specific.

- Sub-step 1: If specific data for a given store is available, attempt automatic string matching on the training data of this selected store.
- Sub-step 2: Attempt automatic string matching on the full data.
- Sub-step 3: If specific data for a given store is available, apply the store-specific machine learning model to find the highest scoring COICOP match.
- Sub-step 4: Apply the machine learning model trained on the full data to find the highest scoring COICOP match.
- Sub-step 5: Allow the user to manually search within the full data, with the option to alter the search string.

If none of these sub-steps yields a satisfactory result, the product row must be manually coded during post-processing within the NSI.

Current state of work and outlook

WP3 is currently developing and testing the different sub-steps using Dutch and German data. CBS is mainly involved in setting up the machine learning based sub-steps, while Destatis is developing different automatic string matching approaches. To quantify the success of each sub-step, several analyses have been performed.

The annotation of scanned receipts is the first way of quantifying the success of some sub-steps. Destatis is currently annotating multiple thousand receipts to create a sufficiently large test corpus. The goal of this annotation task is first, and foremost, carried out to build a test corpus that can be used as a means to validate and test the developed classification pipeline. The classification pipeline is a combination of two modules, the OCR microservice and the COICOP classification microservice. Errors from the OCR microservice may influence the success of the COICOP classification microservice. Collected receipts help to estimate to what extent this may be the case. However, the annotated receipts can be in addition be used as training material for the OCR microservice, in particular for the training of the OCR and document understanding models. Moreover, scanned receipts can be used as (additional) training data for the COICOP classification.

Analyzing product inventory and dynamics of the available store data, is the second way of quantifying the success of some sub-steps. Both CBS and Destatis have analyzed and compared the product inventory of different supermarkets. On the one hand, the analysis was aimed at identifying the overlap (and difference) between the product inventories of the supermarkets. This overlap was measured using product identifiers used by the supermarkets, like for example GTINs, and using the receipt texts directly. The main purpose of analyzing product inventory was estimating the possibility of using data from one supermarket for COICOP classifications of products from another supermarket. On the other hand, the analysis was aimed at the product dynamics of a store. Product dynamics describe the development of a store's product inventory over time. Analyzing the product dynamics therefore gives an impression of how well a classification method will keep performing over time. Additionally, it will give an indication of how fast the corpus of receipt texts will need to be updated.

Analyzing classifier performance is the third way of quantifying the success of some sub-steps. For the string matching approach this means evaluating its performance using a corpus of scanned receipts. Very preliminary results for a single supermarket in Germany indicates a success rate of around 70%. Such analyses will be intensified in the upcoming months. For the machine learning approach this means evaluating the machine learning pipeline for its generalizing capabilities. This means the evaluation and comparison of different feature extraction methods and machine learning models on chosen performance metrics. In detail, this means measuring performance on a hold-out dataset of data to estimate generalization to unseen data. In addition, it means measuring performance from period to period to estimate real-world performance and model performance decay. Moreover, cross-store performance is another aspect of performance that estimates performance on store data that is not part of the training data. For the manual search approach the quantification is difficult given that the respondent can choose between different offered COICOP categories. Preliminary data from the national HBS 2023 in Germany suggests that respondents

using a well-maintained search algorithm are able to classify 87% to the targeted COICOP category. The remaining 13% are placed into a miscellaneous category and are coded during post-processing by Destatis. Last, the chain of the OCR microservice and the classification microservice should be tested to see how errors propagated from the OCR microservice affect the classification performance of the classification microservice. CBS has already been working on several of these analyses.

Implementation

Hbits has developed the technical solution to host both Python as well as R classification techniques for the classification task within the microservice.

Integration

[TODO next DL]

Test

[TODO next DL]

7. Functional and non-functional requirements of the GeoService Microservice

Just like in chapter 4, chapter 7 has the focus on better involving and engaging households and citizens by defining and operationalizing a new/modified end-to-end data collection process. This time the focus is on making use of geolocation points to support the Time Use Survey.

To collect these geolocation points the use of internal sensors of smart devices is needed. NSIs and linked organizations have worked on platforms to allow households to register their time spending in an online diary. The past few years a multitude of applications were developed to collect time use data, and those related to the ESS have developed these applications in light of the HETUS guidelines. In the SSI project the CBS, hbits, Insee and SSB represent this focus on official (time use) statistics.

The general idea is to provide to the users/respondents a framework of places (stops) and travels (tracks) and the mode of transportation in order to support them in keeping their timeline up-to-date. Within WP3 the GeoService Microservice is developed as middle part software that processes the sensory data in order to provide tentative input to the timeline of the diary.

The main models are based on Stop-Track prediction, on Transport Mode prediction, and on the connection of tracks to transport motivations following the HETUS guidelines to gain TUS relevant information.

Business requirements

TUS gathers information on the daily activities' household members perform. Typical to a TUS is that these activities are being collected with their temporal, spatial and social context. TUS is harmonized via the HETUS-guidelines with the first edition being published in 2000, and recently received its third update with the 2018-guidelines. Member States have the option to collect TUS data. Currently the third data collection is running. Methodological variations between countries apply. In 2030 TUS will enter the IESS agreement on an optional level.

Just like HBS is TUS a household study. In TUS all eligible household members are invited to keep a 2-day diary, on the same moment to be able to study the intra-household allocation of time. Following the HETUS-guidelines, one diary day contains 24 hours running from 4 am until 4 am the next day. Each activity is reported verbatim, both for the main and (possible) parallel activity. The same counts for information on the location or the mode of transport. The use of an electronic device is answered with a tick-box (yes when checked). The social environment is also captured with through tick-boxes collecting information on whether the activity was done alone or together with someone known. A distinction is made between social partners within (partner, parent, child up to 17 years old, other household member) and outside the household. A new episode starts when either an activity and/or one of the contexts change. Every diary day is ended with a small questionnaire asking about the level of satisfaction during the reported day.

Under the wings of the process of modernization, and also under the auspices of EUROSTAT, TUS underwent a mode shift to an online data collection strategy making use of web and mobile supported applications to collect time use data. Initiatives were taken by various Member States and

are inventoried on the EUROSTATS' wiki page:

<https://webgate.ec.europa.eu/fpfis/wikis/display/ISTLCS/TUS+TOOLS+MENU>.

Taking all developments into account, one of the main thresholds for TUS comes from the detailed reporting of activities in a the time-space framework. An important indicator to picture this threshold is the time between the actual action and the reporting of the activity. Studies show that the quality of reporting remains good upon a reporting delay of at maximum 24 hours (see Yesterday reporting). It is however expected that the burden to reconstruct the day turns higher the longer the actual activity has been performed. Depending the detail of the activity (i.e. more activities on the detailed level) an extra impact expected.

The goal of WP3 is to reduce these gaps by developing and implementing microservices that acquire, process and (can) combine data collected from smart devices and other applications, in the case of TUS through the development of a geolocation microservice that through sensor activation captures geolocation points to derive information on the trip, mode of transport and the stops. Related to the stops, extra context can be added through the connection of third-party databases, and a classification algorithm would be able to link a HETUS code activity (or list of activities) to the stop.

A successful realization of the development and implementation will not entirely reduce the active participation of household members in the registration of their daily activities and context, but will provide support and guidance in their task to arrive to qualitative and comparable data for the ESS. It means that besides the development of the microservice also the implementation of the service to the platforms is important, as well as the UI/UX that presents the output of the microservice to the user, and the easiness in which the user can verify, adapt, or even delete the output.

This project will focus on the smartphone as (1) the device to install the mobile application on, and used by the user as interface to partake to the study, as well as (2) being the motion tracker to collect the movements of the respondent/user, as a proxy of the person itself.

The following objectives are essential in reaching this goal:

- Objective 1: To define an architecture of a microservice (that is also to be reused in the other development of WP3)
- Objective 2: To develop a geolocation microservice to predict trips and stops
- Objective 3: To implement classification solutions (machine learning, string matching, or search algorithm based) to classify stop to the HETUS-classification
- Objective 4: To develop an API to connect to/from other environments
- Objective 5: To deploy the microservice as a containerized application in the cloud
- Objective 6: To implement/integrate specific microservice parts in the app (e.g. algorithm). This integration should be feasible, should have an added value for the platform and/or should improve the user experience.

The stakeholders are the NSIs and their product owners, and the households (citizens).

TUS study

In this section TUS studies are being described as they provide the context in which the geolocation microservice operates.

In TUS studies questionnaires and a time diary are completed by the households. At the moment household members arrive to the diary phase they, at the least, already have completed a questionnaire. If this member is the reference person, or the head of the household also a household questionnaire and a matrix to compose the household is part of the pre-diary tasks. All tasks are defined in a respondent journey or study flow that shows a sequence of tasks. Since the TUS diary setup requires an equal distribution of participation over the entire fieldwork period, and household members are requested to keep their diaries for the same period this study flow can be quite complex.

Central to a TUS study is the registration of activities in a diary. All eligible household members keep a diary for the same 2 days, one weekday and one weekend day. Left aside paper-and-pencil diaries, a household member partakes to a TUS study via an application, be it via a mobile application, be it via a web application running in a browser.

TUS diary

The diary collects at the minimum episode information, where an episode is defined by a beginning and ending time and a change of:

- A main activity as defined in the HETUS Activity Classification List (ACL)
- (If any) a secondary activity as defined in the HETUS Activity Classification List (ACL)
- The place of the activity or a mode of transport when moving
- The use of an electronic device, and
- The social context

The registration of the products and services is linked to the HETUS Activity Classification List (ACL), and is demanded to be delivered on 3 digits. NSIs often use more digits to aggregate to a higher level. The HETUS guidelines further describes the other contexts: place 2 digits, electronic device 1 digit, and social context 1 digit.

In addition, extra context can be added to the online diary, an example is motivation.

Every diary day collects extra information through a small questionnaire, it relates to:

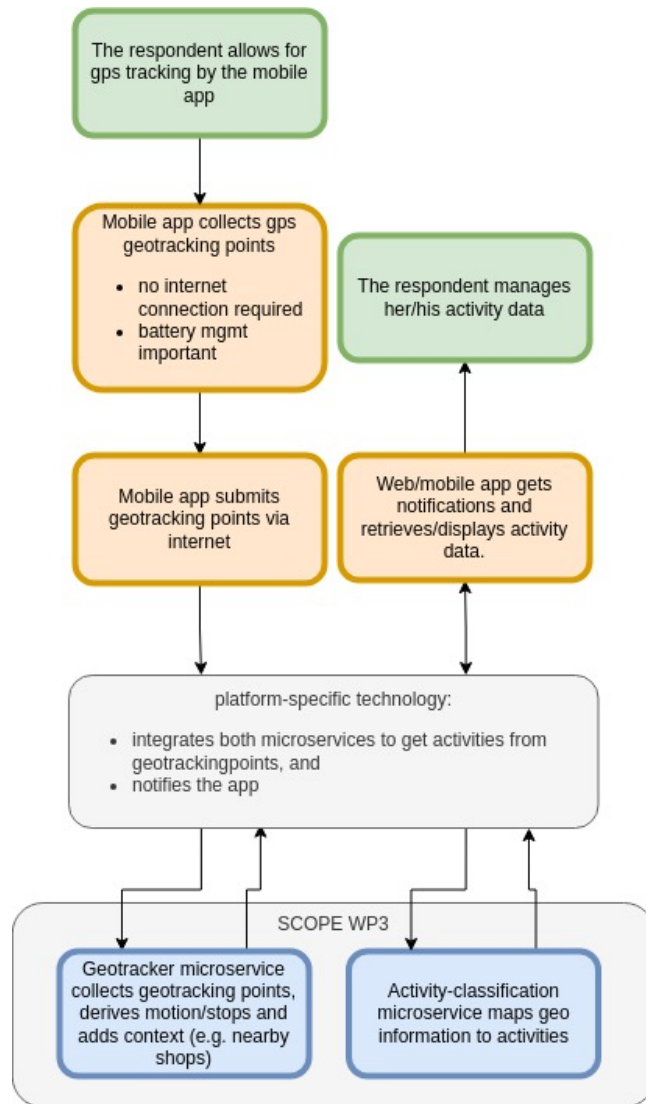
- When the diary was completed
- What the most pleasant activity was
- What the most unpleasant activity was
- What the most stressful activity was
- The overall appreciation of the day
- Whether the day was ordinary or unusual
- Whether a trip within the country or abroad was made, and how far the trip was

Functional requirements

The diagram gives an overview of the main functional requirements:

- functionality related to *user handling* is indicated by the green boxes. The respondent must be able to switch on the sensors to track the movement of the smartphone. After processing by the microservices, the respondent can view/edit/manage her or his activities.

- functionality related to the *app* is indicated by the yellow boxes. The app is responsible for GPS tracking, display of information and communication with the platform core.
- functionality related to the *microservices* is indicated by the blue boxes. The function is to derive essential information on trips, mode of transport, stops, context of stops and activity classification. Only the blue boxes are in scope of WP3.



User handling

Respondent handling (green boxes)	
REQ R1	The respondent allows for GPS tracking by the mobile app
	Technically, the mobile phone OS needs permission to enable GPS tracking for the mobile app.
	Formally, a user consent is required.

REQ R2	The respondent manages her/his activity data
	A possibility (~MOTUS) is to present the activity list to the respondent as tentative data. The respondent is then able to edit this list before it becomes final.

App

App (yellow boxes)	
REQ A1	Mobile app collects GPS geotracking points
	Does not require an internet connection.
	Battery management is important: a balance between battery lifetime and frequency of measurements.
REQ A2	Mobile app submits geotracking points via internet to the platform core
	Battery management: a balance between battery lifetime and frequency of sending results to the platform core.
REQ A3	App gets notifications and retrieves/displays activity data.
	App and platform core communicate with each other to exchange activity data. The data has been processed by the platform and its microservices.

Microservice

Microservice (blue boxes)	
REQ G1	Geotracker microservice collects geotracking points
	Geotracker (regularly) receives (new) tracking points from the platform.
	An internal database stores all tracking points
REQ G2	Geotracker microservice derives motion/stop
	An algorithm processes the tracking points in order to find a timeline of motions (transport) and stops.
	The algorithm could use external sources such as openstreetmap in order to improve the results.
	Support for user-specific locations (home, work etc.) is required as well.
REQ G3	Geotracker microservice adds context to motion/stops
	External sources can be consulted to add context. E.g. for stops, a list of nearby places/shops could be added.
REQ A1	Activity microservice assigns scores to POIs
	A score (POI-score) is assigned to each POI inside an adaptive radius around the stop centre location, based on the weighted median of the distances calculated between each POI and all GPS points of the stop, weighting by the accuracy of GPS points.
	A short list of POIs is identified using the elbow criterion on the POI scores. <i>Categories of place are assigned to each POI</i>
REQ A2	Activity microservice associates activities to categories of places
	Through a Bayesian decomposition, for each POI of the short list the conditional probability of HETUS activities are calculated starting from the distribution observed in TUS data. The variables considered (duration and time of the day, HETUS place category, occupational status, age classes) in the decomposition are linked with the corresponding variables observed in the stop and for the specific respondent.
	A rank of the HETUS activities is assigned to the stop, based on a final score calculated aggregating the probabilities of the activity weighted by the POI-score associated with the activity for each POI in the shortlist.

Non-functional requirements

Non-functional requirements	
REQ N1	A microservice should be independent from any specific HBS platform.
	A microservice has no dependency to other environments, and has an independent operation.
REQ N2	It must be possible to connect and communicate with a microservice from any HBS platform.
	A microservice receives input, and provides output making use of APIs.
REQ N3	A microservice must have a design in which algorithms (computer vision, AI, ML) can be easily improved/updated.
REQ N4	The service must be deployable at any institute/NSI (shareability).
	Microservices are provided as software packages in containers, which can be easily shared and deployed. Docker is a software that can host containers. Kubernetes is often used as software to orchestrate various containers.
REQ N5	The service must be scalable with the number of receipts it needs to handle.
	Kubernetes is a software used to orchestrate containers. By this Kubernetes allows to horizontally scale the containerised microservice depending to the number of receipts received.
REQ N6	Security by design
	Using the container technology barriers are created between various components used in the study setup, which deliver better privacy, security and maintainability, scalability and high availability. Communication between the platforms runs through APIs and https communication.
REQ N7	Privacy by design
	Using the container technology barriers are created between various components used in the study setup, which deliver better privacy, security and maintainability, scalability and high availability. Communication between the platforms runs through APIs and via UUIDs to avoid transferring personal information.

REQ N8	Support for localization
	Algorithms being applied by the microservice should be configurable or trainable (in case of ML) to support localization, which includes different languages, different currencies, date formats, dots vs commas etc. This is required to make the microservice shareable.
REQ N9	Offline vs online support (app)
	Parts of the microservice are/can be selected to be developed in a Library to run offline in an application. The library must take into account platform-dependency (Angular, ionic, Flutter ...) to function.

8. Geolocation microservice determining stop-track clusters (GeoService Microservice – part 1)

The GeoService Microservice is supported by two microservices: the Geolocation Microservice (part 1) and the HETUS classification Microservice (part 2).

The design of the Geolocation Microservice holds two important elements: the definition of the stop-track clusters and the prediction of the travel mode upon the track clusters. These elements will be discussed in 2 chapters but finally result in one microservice, the Geolocation Microservice.

This chapter discusses the algorithm to derive the stop and track clusters.

Design

After the Geolocation Microservice gets the geolocation points the stop-track part starts with the stop detection algorithm which takes into account 4 steps:

- Filter GPS points based on accuracy,
- Determine which GPS points are significant stop points
- Cluster the stop points, and
- Post-processing
 - reduce number of clusters (merging)
 - guarantee stops and tracks alternately

The required input parameters are timestamp, longitude, latitude and accuracy. The list of geolocation points needs to be time-ordered.

In an extra step, also, extra information can be attached to the stop clusters by connecting to a POI or places API:

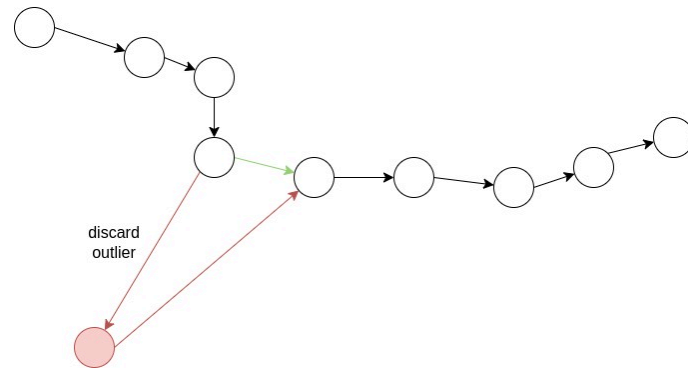
- Point-of-interest: find places inside or nearby stop clusters (e.g. OpenStreetMap, Google Places)

The output of the algorithm is a list of alternating stop and track clusters. Stop clusters will have extra information regarding nearby places (e.g. shops).

Pre-processing

Filter GPS points

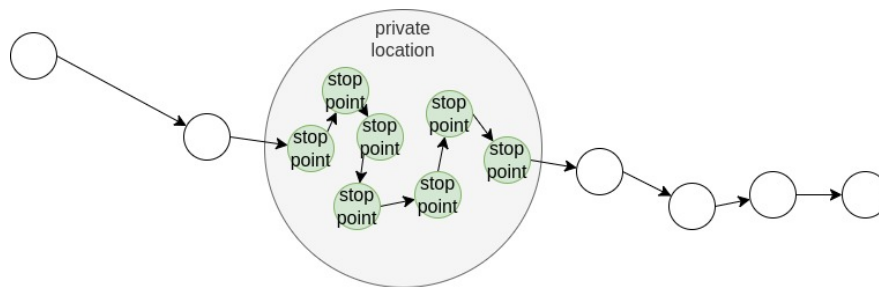
Filter GPS point which has good enough accuracy. Currently, the algorithm requires an accuracy of 100m, but this value is configurable.



Determine significant stops and clusters

Private locations

When a GPS point falls in a private location (home, work etc.) then it is always regarded as a stop. A private location is defined as a circle (lon, lat, radius).



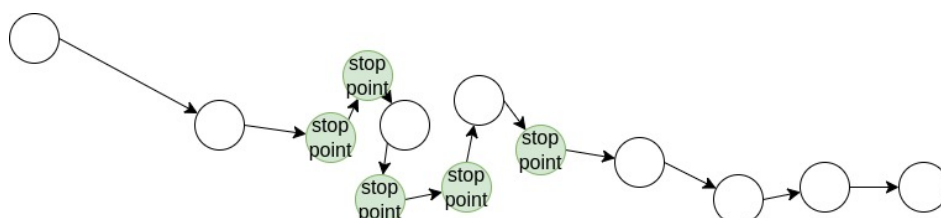
ATS to determine significant stops

The algorithm for stop detection is implemented from the paper “Individual and collective stop-based adaptive trajectory segmentation” from Agnese Bonavita, Riccardo Guidotti and Mirco Nanni, as published in *Geoinformation* (2022) 26:451-477. From this paper, only the individual stop-based adaptive trajectory segmentation (ATS) has been implemented.

Essentially, the algorithm decides that a GPS point is a stop point when more than t seconds is spent between the current GPS point and the next GPS point that is more than x meters away.

The algorithm can be tuned by changing the temporal and/or spatial parameters. By default, the implementation used 50m as spatial parameter (as advised in the paper) and 180s for the temporal parameter.

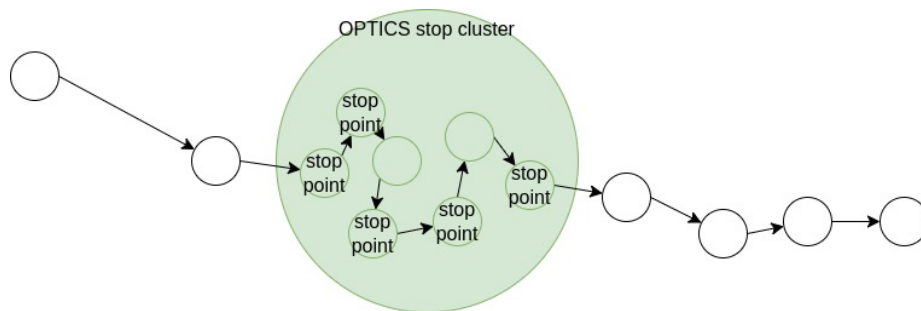
Note that deriving the temporal parameter from the GPS data by means of a Thompson tau statistic is also supported by the algorithm. The description can be found in the paper.



Cluster stop points

Once the stop points have been determined, the next step is to cluster the stop point in order to get stop clusters. In order to achieve this, the project decided to use the well-known OPTICS algorithm (derived from DBSCAN), which applies a density-based technique on spatial data (https://en.wikipedia.org/wiki/OPTICS_algorithm).

Because OPTICS does not take into account the time aspect of the data, the results are further processed to split the spatial clusters based on time as well.



Post-processing

Finally, in order to deliver a clean output, merging of stop clusters is done after which track clusters are added between the stops.

Point-Of-Interest (POI)

[TODO next DL Google Places]

[TODO next DL OSM]

Implementation

An R implementation with bindings to C++ is made and available in git. Currently, the private locations and nearby places are not implemented yet.

Integration

[TODO next DL]

Test

[TODO next DL]

9. Geolocation microservice detecting the mode of transport (GeoService Microservice – part 1)

The GeoService Microservice is supported by two microservices: the Geolocation Microservice (part 1) and the HETUS classification Microservice (part 2).

The design of the Geolocation Microservice holds two important elements: the definition of the stop-track clusters and the prediction of the travel mode upon the track clusters. These elements will be discussed in 2 chapters but finally result in one microservice, the Geolocation Microservice.

This chapter discusses the detection of the mode of transport upon the track clusters.

Design

After classifying the geolocations into stop and track clusters, the track clusters are fed into the transport mode prediction algorithm. The main principle behind the described method is the match of these track clusters to the infrastructure of a range of transport modes. The more points within a cluster match with a transport mode, the higher the proportion that is assigned to this transport mode.

Input data

The algorithm is written in R and required the following input data.

Classified clusters

The algorithm requires the clustered geolocations of the users in the following format:

user_id	cluster_id	timestamp	coordinate	accuracy	classification
integer	integer	(POSIXct)	(lat, lon)	integer	track stop

An example data set is:

user_id	cluster_id	timestamp	coordinate	accuracy	classification
1	1	28-03-2024 – 09:43:01	(lat, lon)	1	Stop
1	1	28-03-2024 – 09:43:03	(lat, lon)	2	Stop
1	1	28-03-2024 – 09:43:05	(lat, lon)	3	Stop
1	1	28-03-2024 – 09:48:01	(lat, lon)	4	Stop
1	2	28-03-2024 – 09:48:05	(lat, lon)	5	Track
1	2	28-03-2024 – 09:52:01	(lat, lon)	6	Track
1	2	28-03-2024 – 09:55:01	(lat, lon)	7	Track
1	2	28-03-2024 – 09:57:01	(lat, lon)	8	Track
1	2	28-03-2024 – 09:58:01	(lat, lon)	9	Track
2	1	29-03-2024 – 09:42:04	(lat, lon)	10	Track
2	1	29-03-2024 – 10:42:05	(lat, lon)	11	Track
2	1	29-03-2024 – 10:42:07	(lat, lon)	12	Track
2	1	29-03-2024 – 10:45:04	(lat, lon)	13	Track
2	1	29-03-2024 – 10:48:08	(lat, lon)	14	Track
2	2	29-03-2024 – 10:49:01	(lat, lon)	15	Stop
2	2	29-03-2024 – 10:55:04	(lat, lon)	16	Stop
2	2	29-03-2024 – 10:59:01	(lat, lon)	17	Stop

Travel mode data points

The transport modes used are: motorized vehicles on roads, train, tram, bus, subway, bicycle, by foot. For each mode and for a specified region (e.g. the Netherlands), a dataset is extracted from OSM. This is a static copy of OSM that needs to be updated at a certain frequency in time. The number of subway and tram location points is relatively small, whereas roads, cycling paths and sidewalks are very large.

The algorithm

For each track cluster, the algorithm performs the following steps:

1. A geographical box (parameterized and could be changed) is made around each users' geolocation
2. A sub selection of all travel mode data points that fall within this geographical box is made in order to reduce the dataset and thus computation times.
3. Per transport mode:
 - a. The minimum distance between the geolocation of the user and the travel mode data points in the sub selection is determined.
 - b. It is verified whether the minimum distance is smaller than the accuracy of the user geolocation
4. Steps 1, 2 and 3 are performed iteratively for each individual data point of each user.
5. The number of location data points that are identified in Step 3b are counted separately for each transport mode.
6. The number of points per transport mode is divided by the total number of points in the cluster to get proportions for each transport mode.
7. The transport mode with the highest proportion is chosen.

This results in one or multiple transport modes for each track cluster. A few remarks are in place:

- The method will return proportions for all travel modes in the cluster. The travel mode with the highest proportion is selected as transport mode for the cluster, independent on the height of the proportion. Even if only very few user geolocations could be linked to one of the transport mode, this mode could become the selected mode.
- Currently, there is no implemented strategy on how to handle equal proportions of clusters. Both transport modes are returned.
- The method assumes that each cluster is unimodal. No strategy is implemented to determine whether multiple transport modes were used within one track cluster. Therefore, the proposed method quickly loses accuracy when a cluster consists of different travel modes.
- The performance will highly depend on the quality, in particular the density, of the dataset per transport mode.
- The travel mode prediction was developed on data gathered a specific sensor configuration, assuming an ongoing data stream of high-frequent geolocation data, and a specific clustering algorithm, resulting in relatively small track clusters. Applying the algorithm on data gathered with a different sensor configuration and clustered using a different clustering algorithm is likely to reduce validity and performance.

Implementation

[TODO next DL, the algorithm is currently implemented in R]

Integration

[TODO next DL]

Test

[TODO next DL]

10. HETUS classification Microservice (GeoService Microservice – part 2)

The GeoService Microservice is supported by two microservices: the Geolocation Microservice (part 1) and the HETUS classification Microservice (part 2).

The second part of the GeoService Microservices aims to provide information about the most likely activity carried out by the user during a stop. For the activity taxonomy, we refer to HETUS, "Harmonised European Time Use Survey". To predict the activity performed by the user at a stop, we can leverage the spatiotemporal characteristics of the stop, the types of points of interest near the stop such as restaurants, shops, cinemas, schools, theaters; the socio-demographic characteristics of the user like age group or employment status.

These important pieces of information (spatiotemporal, description of nearby points of interest, and user characteristics) are the input of the algorithm and must be provided in the request to the service, specifically the spatiotemporal characteristics of the stop and the list of nearby points of interest, which are produced in the Geolocation Microservice part 1 described in Chapter 8.

Since we cannot directly have a labeled dataset that associates the input variables described (X) with the activity carried out according to the HETUS classification (Y), the algorithm underlying this microservice exploits the data collected in the TUS surveys of national statistical institutes.

Design

We describe the Activity Classification of a stop as detected by the ATS_OPTICS algorithm.

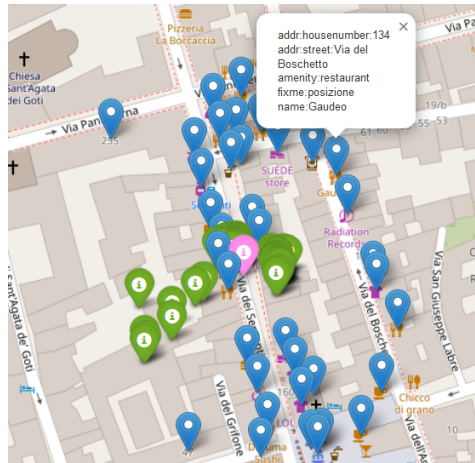
Input Structure Description

For a stop, the algorithm expects an input consisting of a data structure with:

- For each GPS points of the stop:
 - o GPS Longitude
 - o GPS Latitude
 - o GPS Accuracy
 - o GPS Timestamp
- For each Point of Interest (POI) inside the radius of the stop:
 - o POI Longitude
 - o POI Latitude
 - o Tag (textual description provided by MapService)
- Profile of the user
 - o Age class
 - o Condition (employed, student, other)

The textual description depends on the map service used. In Google Maps, we use the name and type of the place. In Open Street Maps, the name of the place and the values of tags such as Amenity, Shop, Office, Leisure, etc., are used. Depending on the research and privacy rules, it may not be possible to use the respondent's information (age class, condition). In this case, the prediction will be made with the other available information. The impact of this lack of information on the quality of the HETUS activity prevision will be measured in subsequent assessment tests.

The start time, end time, duration, and centroid of a stop are calculated from the set of GPS points of the stop.



In the figure above, we show some information about the input to the microservice. The blue points are the POIs around the stop, the green points are a sample of the GPS points belonging to the stop, and the pink point is the centroid of the stop. For a POI, we display the tag provided by the OSM map service.

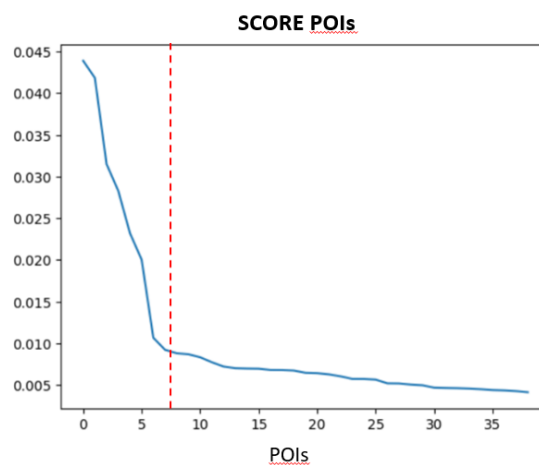
Shortlist of Points of Interest

In this initial phase, the goal is to have a shortlist of the most probable points of interest where the user is likely to have carried out their activity. This is a challenging task due to the uncertainty of the GPS sensor and errors in segmenting the GPS points.

For each point of interest, we calculate a score based on the median distance of the POI from the GPS points belonging to the stop, weighted by the accuracy, according to the following formula:

$$ScorePOI_i = median_j^{N-GPS} \left(\frac{1}{distance(GPS_j, POI_i) accuracy_j} \right)$$

We then select a shortlist of POIs with the highest POI scores. The selection is made by identifying where the slope of the POI Score curve changes significantly, using the *elbow algorithm*.



Classification of Points of Interest

Each point of interest on the shortlist is classified according to the Italian Time Use Survey classification of the places (TUS place) where the user's activity takes place. The TUS place classification is a specific classification of locations used internally at the statistical institute that provides TUS data. In particular, in the developed prototype, we used data from the Italian TUS survey.

In the following table, we show an extraction from the TUS places table which categorizes a number of 25 places for activities that are more detailed compared to the HETUS classification of places.

TUS place	Description	HETUS Place
04	Hotel and similar	EU17
05	Place of work	EU13
06	School, university	EU13
07	Library	EU19
08	Study center	EU19

The association is made through the search for keywords and regular expressions.

lat	lon	scorePoi	tag	TUS_PLACE
41.895596	12.490317	0.043904	amenity:restaurant name:Temakinho Rome Monti	09
41.895668	12.490338	0.041848	addr:housenumber:130 addr:street:Via dei Serpenti amenity:fast_food name:il Gelatone	09
41.895516	12.490369	0.031499	amenity:restaurant cuisine:regional name:Al Vino al Vino restauranttype:enoteca	09
41.895688	12.490533	0.028297	name shop	12
41.895742	12.490499	0.023218	name name:it shop	12
41.895758	12.490278	0.020034	addr:street:Via dei Serpenti amenity:cafe cuisine:sandwich;coffee_shop internet_access:wlan name:Antico Caffè del Brasile - Torrefazione name:de:Antico Caffè del Brasile - Torrefazione	09
41.895919	12.490231	0.010666	addr:housenumber addr:street delivery name operator phone ref:vatn shop	12

Statistical model to predict stop activity with respect to a POI

For each POI on the shortlist, the probability of performing an activity according to the HETUS classification with respect to a POI is calculated. This probability is decomposed using a Bayesian approach, as outlined in the following formula:

$$P(A_i | x, t) = \frac{N(t, \mu_{x,A_i}, \sigma_{x,A_i})P(x | A_i)P(A_i)}{\sum_j P(x | A_j)P(A_j)}$$

where A_i is the HETUS activity, x is the tuple (user's condition, user's age class, TUS PLACE of the POI) and t is the duration of the stop, while μ_{x,A_i} and σ_{x,A_i} are respectively the mean and standard deviation associated with an activity for a user's condition, user's age class, and TUS PLACE of the POI.

In accordance with the chosen approach, we estimate the model parameters μ_{x,A_i} and σ_{x,A_i} and the probabilities $P(x \vee A_i)$ and $P(A_i)$ using aggregated counts from the TUS survey data.


For clarification, we present an example showing the data in the table with aggregated TUS counts. The table shows that in the TUS survey, the HETUS activity 031 'Washing and dressing' performed by employed respondents (condition=1), aged 15-24 years (age class=1), carried out at their own residence (TUS place=1), was recorded 1837 times with an average duration of 16 minutes and a standard deviation of 12 minutes.

condition	age class	TUS place	HETUS	COUNT	DURM	SD
1	1	1	31	1837	00:16	00:12

Prediction of the stop activity

Finally a rank (ActivityScore) of the HETUS activities is assigned to the stop, based on a final score calculated aggregating the probabilities of the activity weighted by the POI-score associated with the activity for each POI in the shortlist.

ActivityDescr	ActivityScore	PoiScore	HETUS	ActivityProb
021 Eating	0.023086	0.036687	021	0.629271
519 Other or unspecified social life	0.012023	0.036687	519	0.327735
732 Parlour games and play	0.000497	0.036687	732	0.013549
513 Celebrations	0.000387	0.036687	513	0.010555
821 Watching TV, video or DVD	0.000384	0.036687	821	0.010468
522 Theatre and concerts	0.000123	0.036687	522	0.003348
831 Listening to radio or recordings	0.000101	0.036687	831	0.002746
383 Reading, playing and talking with c	0.000048	0.036687	383	0.001320
811 Reading periodicals	0.000026	0.036687	811	0.000702
735 Mobile games (on handheld device/ s	0.000011	0.036687	735	0.000294



HETUS	ActivityScore	Descr
021	7.400402e-02	021 Eating
361	4.739460e-02	361 Shopping (including online/ e-sho
519	3.842740e-02	519 Other or unspecified social life
032	3.437884e-03	032 Personal care servi ces
732	1.588585e-03	732 Parlour games and play
513	1.237589e-03	513 Celebrations
821	1.227424e-03	821 Watching TV, video or DVD
522	3.925254e-04	522 Theatre and concerts
343	3.686005e-04	343 Caring for pets
831	3.219861e-04	831 Listening to radio or recordings
383	1.547464e-04	383 Reading, playing and talking with c
811	8.229526e-05	811 Reading periodicals

this paragraph describe how the algorithm returns to the platform a score related of the most likely HETUS activities given the spatiotemporal characteristics of the stop, the user's characteristics, and the type of POIs around the stop.

Implementation

An implementation in Python has been created and will be available on Git [TODO next DL]. Modifications to the model are planned with the addition of input features useful for identifying the activity, such as the start time of the stop or the day of the week. This implies a modification to the formula for calculating the model based on the aggregated TUS data.

Integration

[TODO next DL]

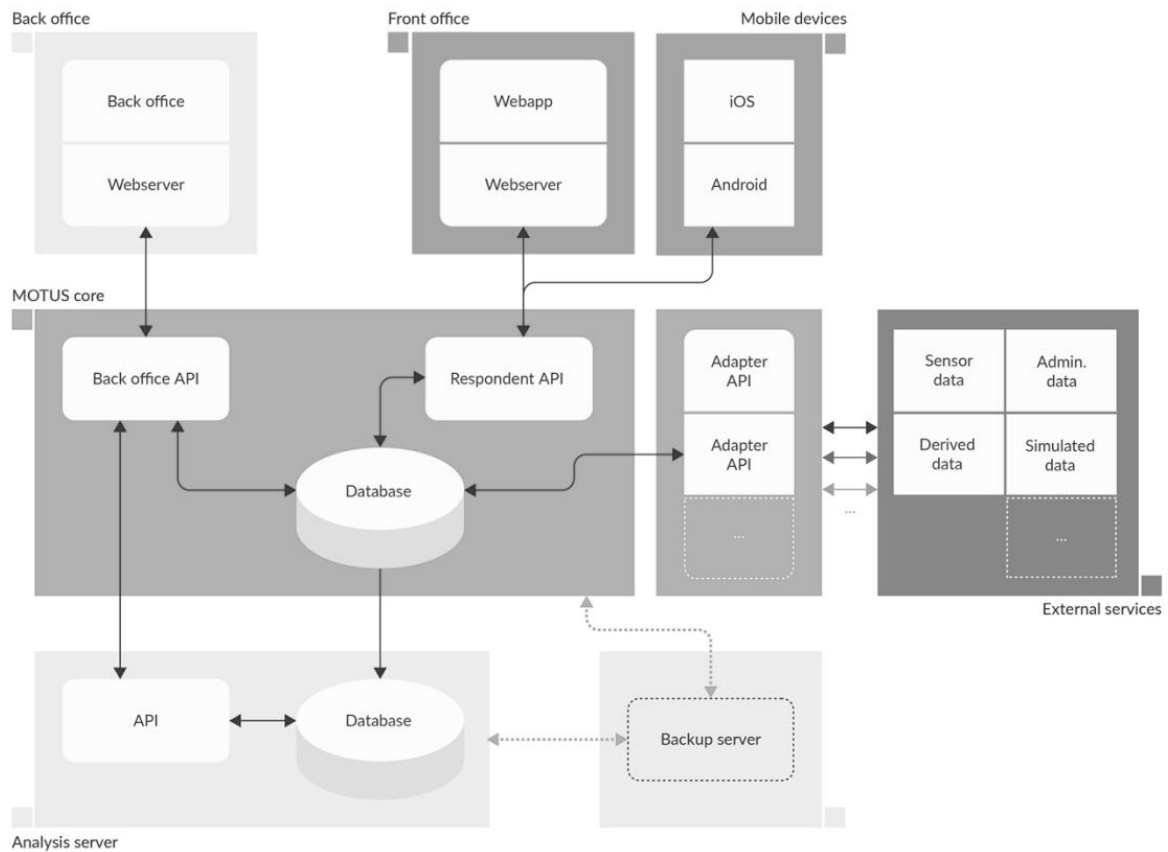
Test

[TODO next DL]

11. MOTUS architecture

The figure shows the platform architecture of MOTUS. The MOTUS data collection platform consists of a front office as well as a back office. The front office relates to the collection tool or application, with which the users can interact via a user interface (UI) and which delivers, through its functionalities, a user experience (UX). The MOTUS application is available as a web version for browsers (<https://app.motusresearch.io>) and in iOS and Android mobile versions for smartphones and tablets. The purpose of the application is to make it easier for the respondent to carry out all tasks of a (time use or other) survey.

The back office serves to build a study, to facilitate data collection and monitoring, and to process the data. To this end, the back office, which is accessible via a web environment, contains several builders. Both the front office and back office connect to the MOTUS core (“the core”) through Application Programming Interfaces (APIs). The core holds the database with all information required to build a study and collect data. A separate analysis server holds a replica of the database from the core and facilitates the processing of information in the back office. The back-up server is a replica of the core and analysis server. Adapter APIs serve to adapt external information so that it can be processed in the core, thereby allowing the ingestion of, for example, passive data gathered via integrated sensors or connected devices, administrative/secondary data available via external data sources, or other processed data. For reasons of optimization, data security and privacy, these data are handled and organised in an anonymized way in stand-alone microservices. All input provided by the user is sent encrypted via an https communication to the server and is immediately propagated to all devices of the user via the respondent API. As a result, the MOTUS web and mobile applications can be used interchangeably.



Although MOTUS can be deployed in various ways, the advice is to follow a cloud-based deployment strategy in which all MOTUS components (core, backoffice...) but microservices as well run in their own container. The benefits of a containerized environment is better scalability, improved application monitoring and decoupling from the underlying infrastructure. Several container management and container orchestration applications exist from commercial to open-source solutions, from simple to complex orchestration platforms. Examples are: docker compose, kubernetes, rancher, redhat openshift.

12. CBS architecture

[TODO next DL]

Glossary

AI	Artificial Intelligence
API	Application Programming Interface
CBS	Centraal Bureau voor de Statistiek, Netherlands
COICOP	Classification Of Individual Consumption According to Purpose
DB	Database
Destatis	Statistisches Bundesamt, Germany
ESS	European Statistical System
hbits	Spin-off company, Belgium
HBS	Household Budget Survey
HETUS	Harmonised European Time Use Survey
INSEE	National Institute of Statistics and Economic Studies
ISTAT	Italian National Institute of Statistics
ML	Machine Learning
MOTUS	Modular Online Time Use Survey
NSI	National Statistical Institute
OCR	Optical Character Recognition
PDCA	Plan Do Check Act
SSB	Statistics Norway
SSI	Smart Survey Implementation
TUS	Time Use Survey
UI	User Interface
UX	User Experience
WP	Work Package