Project number:	693 729
Project acronym:	smarticipate
Project title:	smart open data services and impact assessment for open governance
Instrument: **Call identifier:** **Activity code:**	Horizon2020 H2020-INSO-2015-CNECT INSO-1-2015

Start date of Project:	2016-02-01
Duration:	36 month

Deliverable reference number and title (as in Annex 1):	**D6.1 Integration Requirements Report**
Due date of deliverable (as in Annex 1):	**PM 12**
Actual submission date: **Revision:**	*see "History" Table below*

Organisation name of lead contractor for this deliverable:
GeoVille

Project funded by the European Commission, Horizon2020, topic INSO-1-2015		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium	
CO	Confidential, only for members of the consortium (including the Commission Services)	

European Commission
Information Society and Media

Title:
D6.1 – Integration Requirements Report
Author(s)/Organisation(s):
Norman Kießlich, Maria Lemper, Michel Schwandner, Wolfgang Stemberger (GeoVille)
Working Group:
WP6: GeoVille, Fraunhofer IGD, AIT, WeTransform, UWE, Hamburg, RBKC, Rome
References:

Short Description:
This report specifies the system, hardware, software and data along with integration concept and deployment flowchart.
Keywords:
Deployment, integration, testing

History:				
Version	Author(s)	Status	Comment	Date
001	Maria Lemper, Wolfgang Stemberger	rfc	Table of contents	26.11.2016
002	Norman Kießlich	rfc	Chapters 1,2,3	17.01.2017
003	Maria Lemper, Wolfgang Stemberger	Rfc	Review and input to chapter 2.6	25.01.2017
006	Maria Lemper	Final	Adjustments based on review comments	06.02.2017

Review:			
Version	Reviewer	Comment	Date

004	Nicole Schubbe	Quality Review	30.01.2017
005	Jan Peters-Anders	Quality Review	03.02.2017

FIGURES

TABLES

1 Introduction

The smarticipate project aims to develop ICT tools for participatory applications, which use Open Data and other datasets (e.g. land-use, surveys, etc. which are not in public domain). These applications will enable citizens to co-create, to co-design and to take informed decisions by getting feedback on their innovative participatory applications. Also, citizens will be able to share their ideas and opinions which should enrich existing Open Data.

In this respect, smarticipate’s three pilot cities Rome, Hamburg and London (Royal Borough of Kensington and Chelsea (RBKC)) will actively participate in the development of smarticipate’s applications and data acquisition. The project follows a rigorous development process which begins with the identification of the cities’ needs, gathering of their requirements and the definition of use cases. These use cases aim to accommodate real participatory planning scenarios in these cities where citizen participation is expected/encouraged. The main idea is to allow citizens to visually see the development proposal through the smarticipate applications, make changes and get quick feedback on the proposed changes e.g. whether or not a proposed change is economically feasible or if it is compliant to planning laws or environmental regulations. Furthermore, these proposed changes might be shared within local neighbourhoods with the objective to gather additional suggestions, support, criticism, etc. resulting in generating a lot of opinion based data from citizens.

In order to safeguard the compatibility of the individual components, the appropriate system (hardware) configuration and specification for data readiness (size) according to the system architecture (WP3), the following report will document and consolidate the hardware, software and data specifications prior to the integration of the system. Moreover, it will specify the data to integrate for the individual pilot cases, the system component and data integration procedures as well as the necessary pre-processing steps. The processes will be mapped along a deployment flowchart to coordinate the steps and responsibilities along the implementation of WP6.

2 Integration requirements

2.1 System component specification

2.1.1 Hardware

2.1.1.1 Server for software deployment

In order to guarantee maximum up-time and high Internet bandwidth of the finally developed smarticipate system, it was decided to deploy all developed software on a physical server located in a professional data centre. This poses several advantages:

- Adaptability of hardware setup to reflect changing project needs (no big upfront costs)
- Reduced risk of power outage and damage through fire
- High security standards (e.g. protection against DDoS attacks (Distributed Denial of Service))
- 24/7 professional support in case of hardware issues
- Access to high-performance Internet connection

Standard websites are usually deployed on virtual servers offered by data centres. This is a good choice in case no special software is needed and standard configurations (e.g. Apache webserver, MySQL database, PHP) are sufficient. The needs of the smarticipate system are quite complex and therefore a dedicated root server is the better option. With this physical server the configuration is fully flexible, however putting the burden of the operating system and software installations as well as the maintenance (security fixes etc.) on the customer. For choosing a suitable server we especially considered the demands of 3D applications foreseen in some of the cities. This means either an i7 or Xeon CPU (central processing unit) coupled with at least 16 GB RAM (Random Access Memory) are required. Furthermore data mirroring through a RAID (Redundant Array of Independent Disks) configuration of the hard disks is necessary to avoid data loss due to disk failures. We chose a server of the German data centre company Hetzner, which offers a wide range of virtual and root servers. Hardware specifications of the server are provided in the following table.

Table 1: Hardware specification of the rented root server

Product name	Dedicated Root Server EX40
CPU	
Number	1 socket with 4 cores
Type	Intel® Core™ i7-4770 Quad-Core Haswell incl. Hyper-Threading Technology
RAM	
Number	32 GB
Type	DDR3
Hard disks	
Capacity	**2 x 2 TB**

Type	**SATA 6 Gb/s 7200 rpm**
RAID	**Software-RAID-1**
Network	
Network interface	**1 Gbit/s-Port**
Guaranteed bandwidth	**200 Mbit/s**
Traffic included in monthly price	**30 TB**

In addition to the increased reliability through the hard disk mirroring (RAID-1) dedicated backup space is being rented from Hetzner in order to carry out regular backups. Such backups are necessary in case data deletion happens due to human errors.

2.1.1.2 FTP Server

Basically, all smarticipate applications are planned in such a way that web map services are consumed over the Internet, while traditional data transfers from involved city administration to geodata experts of the smarticipate team shall be avoided. Since web map services are not available for all relevant data, we still use an FTP server for traditional data transfers. For this purpose an already existing server of GeoVille's IT infrastructure is used.

2.1.2 System component integration

2.1.2.1 Overall system architecture

The smarticipate system architecture depicted in Figure 1 is divided in two major blocks: On the left, there is the already existing project website (www.smarticipate.eu) which will be serving as a dissemination hub to spread word about the project and its outcomes. The project webpage is running a WordPress instance and is managed by ICLEI. Later in the project the project homepage will link to the smarticipate demo portal (www.smarticipate.eu/platform) and a repository where the platform software will be available for download.

On the right-hand side, there is the actual smarticipate Platform which is composed of the frontend that exposes the system to the user and facilitates the workflow logic (consecutive execution of logical, atomic steps in a workflow) and the backend that holds the core functionality in the form of micro services as well as the basis for all data to be delivered to the services.

WP6 is primarily concerned with the integration of the frontend and backend components and the testing of their interaction through well-defined interfaces to the individual components.

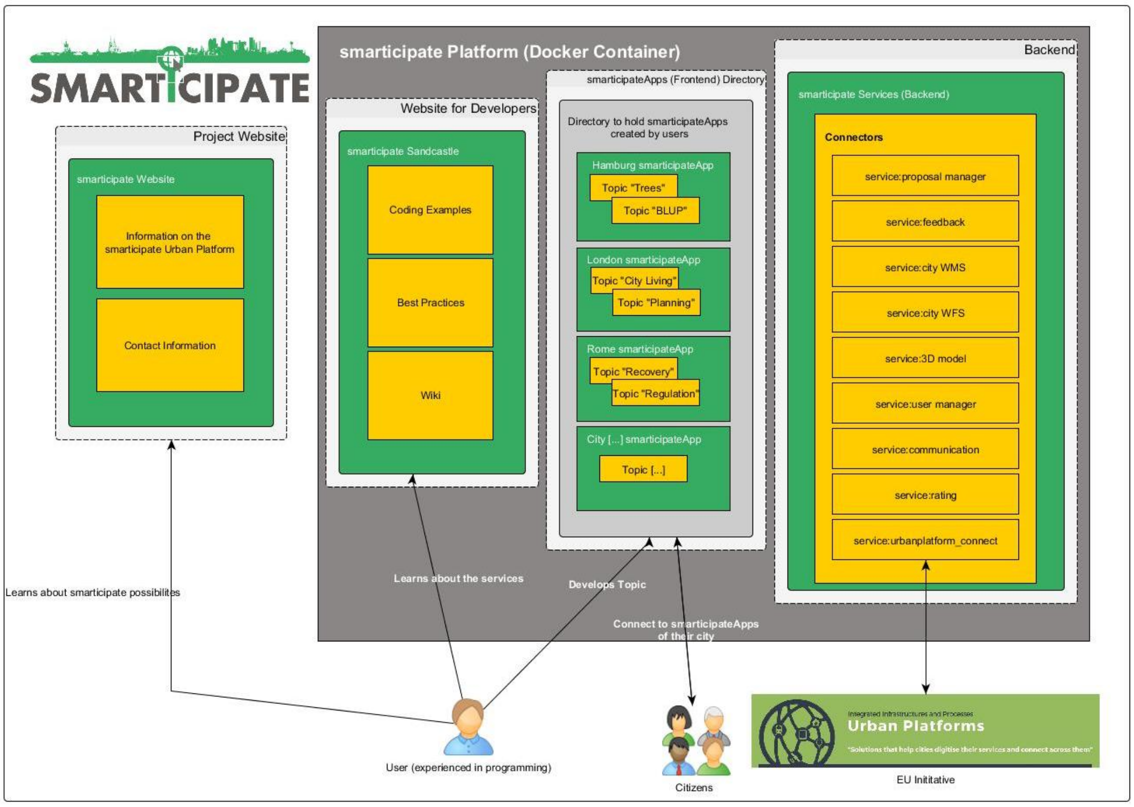


Figure 1: smarticipate's overall system architecture

2.1.2.2 Docker

Due to the fact that the smarticipate platform is a collaborative effort including several developers each with individual preferences and requirements for processing and operating environments, the encapsulation of each micro service into self-contained, runnable containers that only expose an interaction interface is of fundamental importance to the successful integration of all components into a common infrastructure. For this reason, Docker images (Figure 2) are introduced as a state-of-the-art way to deploy such lightweight, self-contained containers of the different components, which can be distributed and used with minimal effort. Using these containers, developers can choose for each component (or service) the technologies best suited for the task, independently of other components. The Docker containers communicate via network interfaces, in most cases based on HTTP. These interfaces shall be RESTful, as described in section 2.1.2.3.

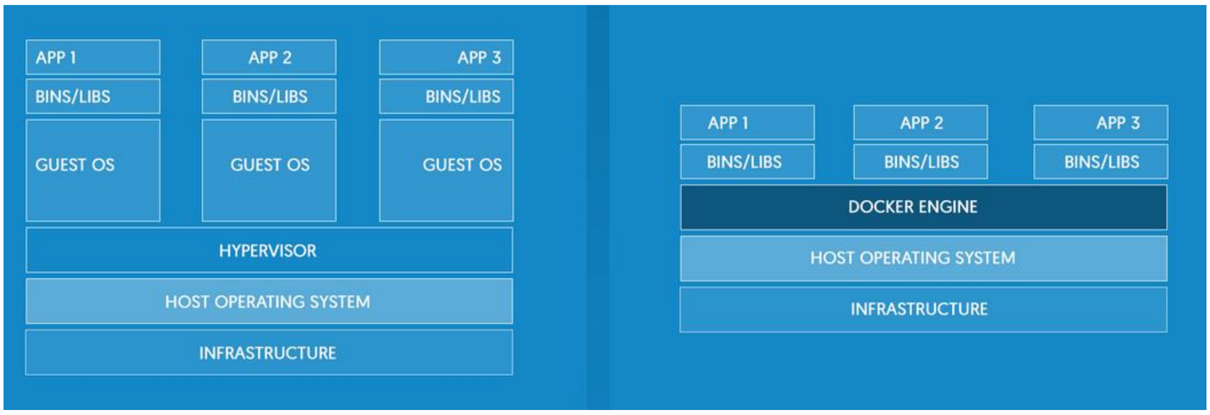


Figure 2: Virtual Machines (left) vs. Docker Images (right) [Source: https://www.docker.com/what-docker]

As outlined in D3.3 (section 2.1.3.2), Docker Compose is proposed as a tool to describe entire whole systems of containers in a simple Docker file format.

The developers of a component/service are required to provide the following along with every submission of a new or updated component:

- Provide a Docker image for their component via a Docker registry, the image should be updated regularly (ideally via an automated build)
- Provide documentation on dependencies (other available Docker images) and configuration options
- Maintain the Docker Compose configuration
- Provide documentation on interfaces intended for interaction with other components (for instance the documentation of a REST API exposed by the service)

Developers only need Docker (https://www.docker.com/) and Docker Compose (https://docs.docker.com/compose/) available on their system to run the whole platform, including the components of their own and other development teams.

The created Docker Compose configuration can also be used for production, often with only a few configuration changes required for the different environment. For a deployment on a single host, Docker Compose can be used directly, for deployment on multiple hosts it can be used for instance in conjunction with Docker Swarm (https://www.docker.com/products/docker-swarm) or Rancher (http://rancher.com/swarm/).

2.1.2.3 Component Interface

In a micro service environment, whole workflows are split into separate steps that are realized as individual micro services. Each service performs a specific task and is minimal in complexity. This requires, however, a standardized communication between all of these services with the frontend application and, potentially, with each other. The interior working of the micro services is not exposed to the outside, rather, they are addressed and respond through RESTful API interfaces. In this way, each micro service wrapped in a Docker container has a well-defined interface to outside clients. The design of the API is dictated by the service function and the requirements of the frontend application requesting its functionality. Thus, it is within the responsibility of the developers (WP4 & WP5) that the RESTful interface of the components corresponds to the needs of the workflows as defined in the Activity Diagrams (see D6.1 section 2.1.2.1). Ideally the interacting components constituting a specific aspect of a scenario are developed simultaneously during a SCRUM sprint and in close coordination between the developers to ensure that the request forwarded by the frontend components are matched with corresponding RESTful API interfaces.

Figure 3 represents the concept of RESTful interfaces to micro services encapsulated in a Docker container.

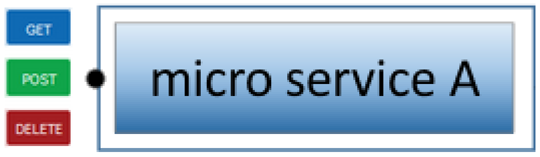
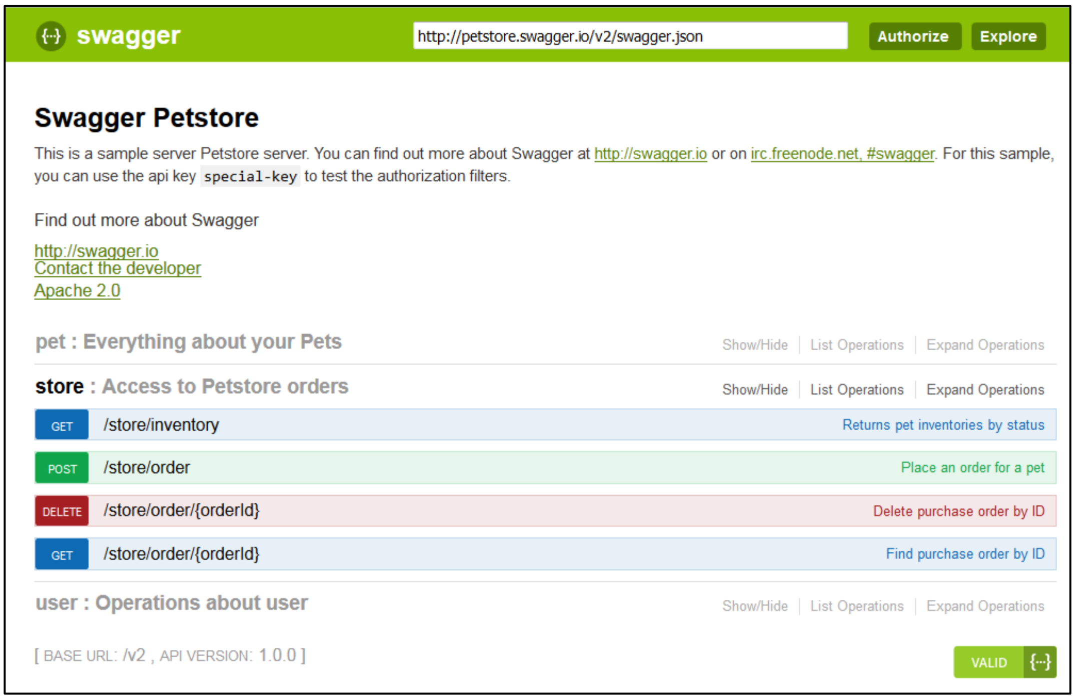


Figure 3: Schematic representation of a micro service providing a RESTful API wrapped in a Docker container.

Complete and accessible documentation of the API is critical in order to ensure that the component is utilized correctly and to its full capability. The API shall therefore be documented using Swagger UI (http://petstore.swagger.io/#/). Swagger UI offers an intuitive and user friendly wiki to a given API. It is automatically generated from Swagger specifications making it easy to implement and maintain on the developers´ side. The visually appealing interface provides a user-friendly overview of the API and also allows for interactive engagement. Figure 4 and 5 provide an example of the Swagger interface listing the methods of a sample API and how such interfaces are to be documented with a detailed example of the methods.



Swagger Petstore

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Find out more about Swagger

http://swagger.io
Contact the developer
Apache 2.0

pet : Everything about your Pets Show/Hide | List Operations | Expand Operations

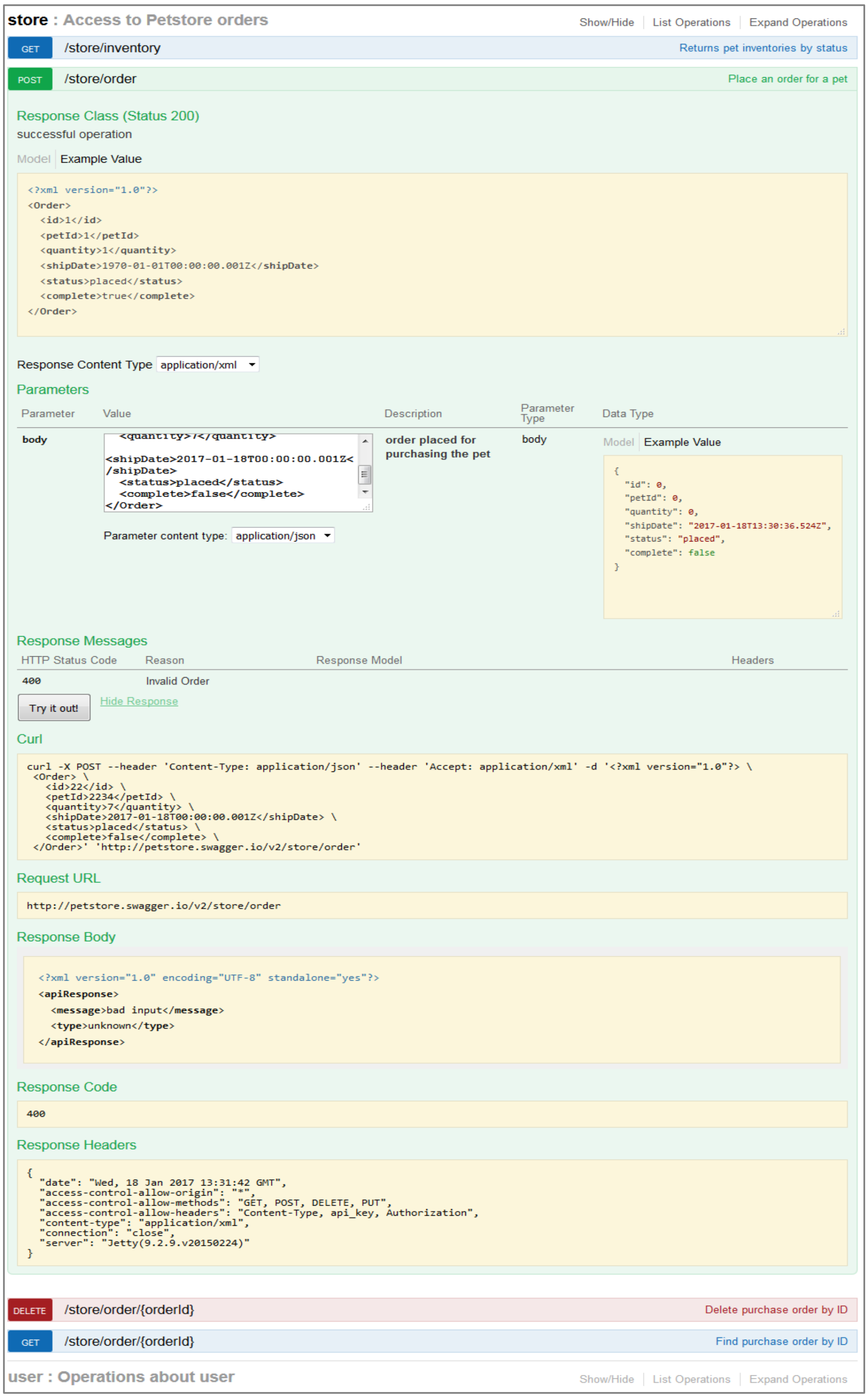
store : Access to Petstore orders Show/Hide | List Operations | Expand Operations

GET	/store/inventory	Returns pet inventories by status
POST	/store/order	Place an order for a pet
DELETE	/store/order/{orderId}	Delete purchase order by ID
GET	/store/order/{orderId}	Find purchase order by ID

user : Operations about user Show/Hide | List Operations | Expand Operations

[BASE URL: /v2 , API VERSION: 1.0.0]

Figure 4: Example of a RESTful API documented with Swagger-UI.



store : Access to Petstore orders — Show/Hide | List Operations | Expand Operations

GET /store/inventory — Returns pet inventories by status

POST /store/order — Place an order for a pet

Response Class (Status 200)

successful operation

Model | Example Value

```
<?xml version="1.0"?>
<Order>
  <id>1</id>
  <petId>1</petId>
  <quantity>1</quantity>
  <shipDate>1970-01-01T00:00:00.001Z</shipDate>
  <status>placed</status>
  <complete>true</complete>
</Order>
```

Response Content Type: application/xml

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	`<quantity>7</quantity>` `<shipDate>2017-01-18T00:00:00.001Z</shipDate>` `<status>placed</status>` `<complete>false</complete>` `</Order>`	order placed for purchasing the pet	body	Model | Example Value

Parameter content type: application/json

```
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2017-01-18T13:30:36.524Z",
  "status": "placed",
  "complete": false
}
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	Invalid Order		

Try it out! Hide Response

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/xml' -d '<?xml version="1.0"?> \
 <Order> \
   <id>22</id> \
   <petId>2234</petId> \
   <quantity>7</quantity> \
   <shipDate>2017-01-18T00:00:00.001Z</shipDate> \
   <status>placed</status> \
   <complete>false</complete> \
 </Order>' 'http://petstore.swagger.io/v2/store/order'
```

Request URL

```
http://petstore.swagger.io/v2/store/order
```

Response Body

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<apiResponse>
  <message>bad input</message>
  <type>unknown</type>
</apiResponse>
```

Response Code

```
400
```

Response Headers

```
{
  "date": "Wed, 18 Jan 2017 13:31:42 GMT",
  "access-control-allow-origin": "*",
  "access-control-allow-methods": "GET, POST, DELETE, PUT",
  "access-control-allow-headers": "Content-Type, api_key, Authorization",
  "content-type": "application/xml",
  "connection": "close",
  "server": "Jetty(9.2.9.v20150224)"
}
```

DELETE /store/order/{orderId} — Delete purchase order by ID

GET /store/order/{orderId} — Find purchase order by ID

user : Operations about user — Show/Hide | List Operations | Expand Operations

Figure 5: Detailed overview of an API method (here: POST) along with interactive testing capabilities.

2.1.2.4 Workflow implementation

The system interaction and communication occurs primarily between the frontend applications and the micro services that in turn access the data storage components (project DB, third party data repositories). Each frontend application (smarticipateApp) hosts a range of topics that in themselves constitute a logical software package for a given use. These scenarios support a workflow of individual steps taken by the end user. It is possible that any given user interaction prompts the front end to request the functionality of a single micro service (Figure 6).

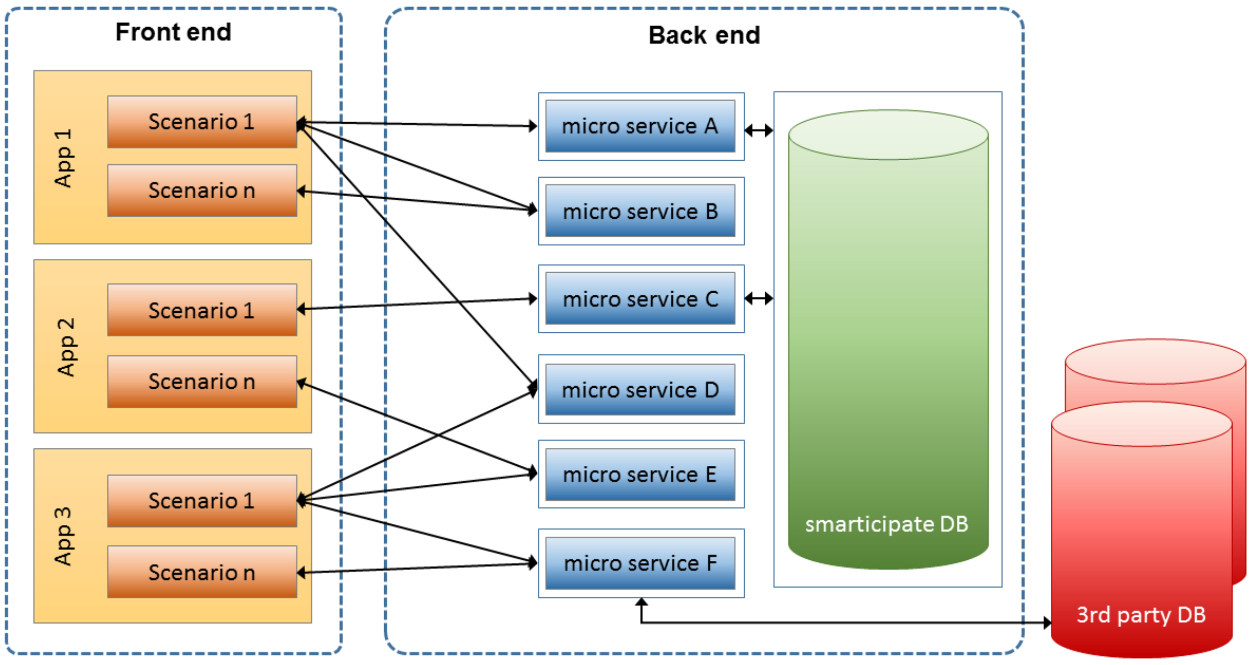


Figure 6: The workflows embedded within the scenarios trigger requests to micro services located in the back end.

However, it is also possible and even likely, that a single step in the frontend user interface triggers a request that needs to be handled sequentially by more than one micro service. The principle of micro services is colliding with the logical grouping of actions anticipated by an end user of a front end. What may seem to be a single processing step to an end user may well be a workflow of two or more micro services to the back end. The principal of intuitive and minimalistic user interaction famously advocated by Steve Jobs[1] in his designs dictates that backend functionality must be hidden from users to the largest possible extent. In contrast, the principle of micro services dictates that services ought to be small and autonomous and not envelope a whole process chain of functions.

In order to accommodate both requirements, workflows must be orchestrated as chains of multiple micro services executed either sequentially or in parallel or as a mix of both. Such a scenario requires either a control component that supervises the execution of this chain of services and reports back the results to the frontend (Figure 7) or an inter-service communication. The latter will be implemented in smarticipate as it is not expected that the micro services are utilized as building blocks for user generated workflows but rather be part of pre-determined and locked workflows (Figure 8). In this case the services that are part of a chain

[1] How Steve Jobs' Love of Simplicity Fueled A Design Revolution | Smithsonian magazine | Sept. 2012 (http://www.smithsonianmag.com/arts-culture/how-steve-jobs-love-of-simplicity-fueled-a-design-revolution-23868877/)

must be aware of each other at least to the extent that they statically call their successor. The absence of a control component has the beneficial effect that it makes the entire system less error prone since a fault in the control component would block the execution of all processing whereas the failure of a single micro service in the latter scenario only affects processing steps that involve that service.

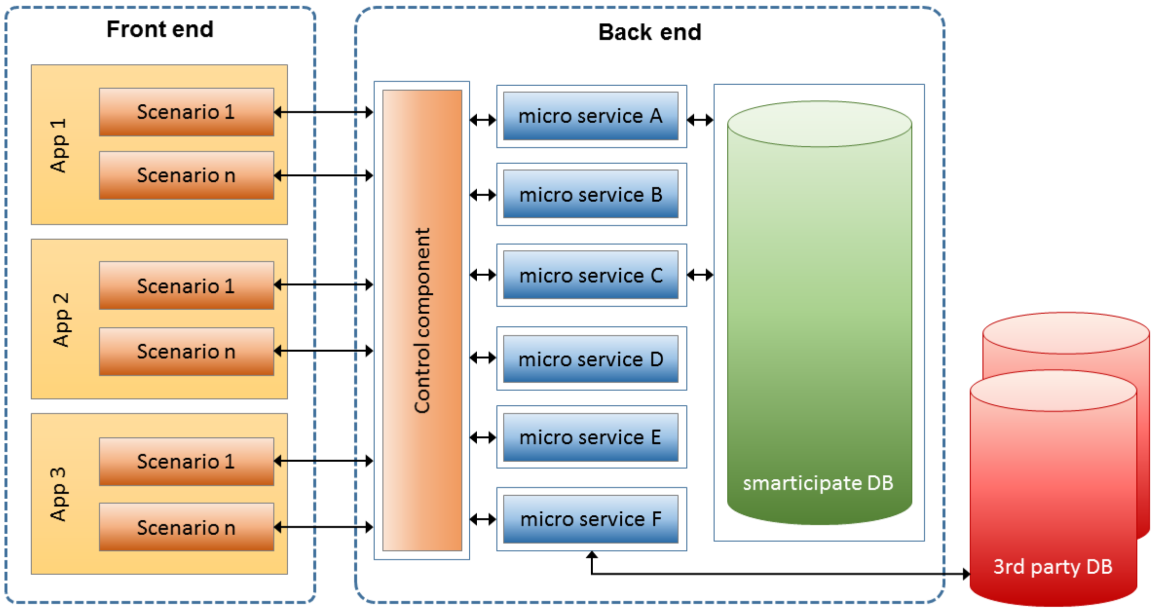


Figure 7: A control component capable of supervising a sequential execution of services is facilitating the communication between front end and back end.

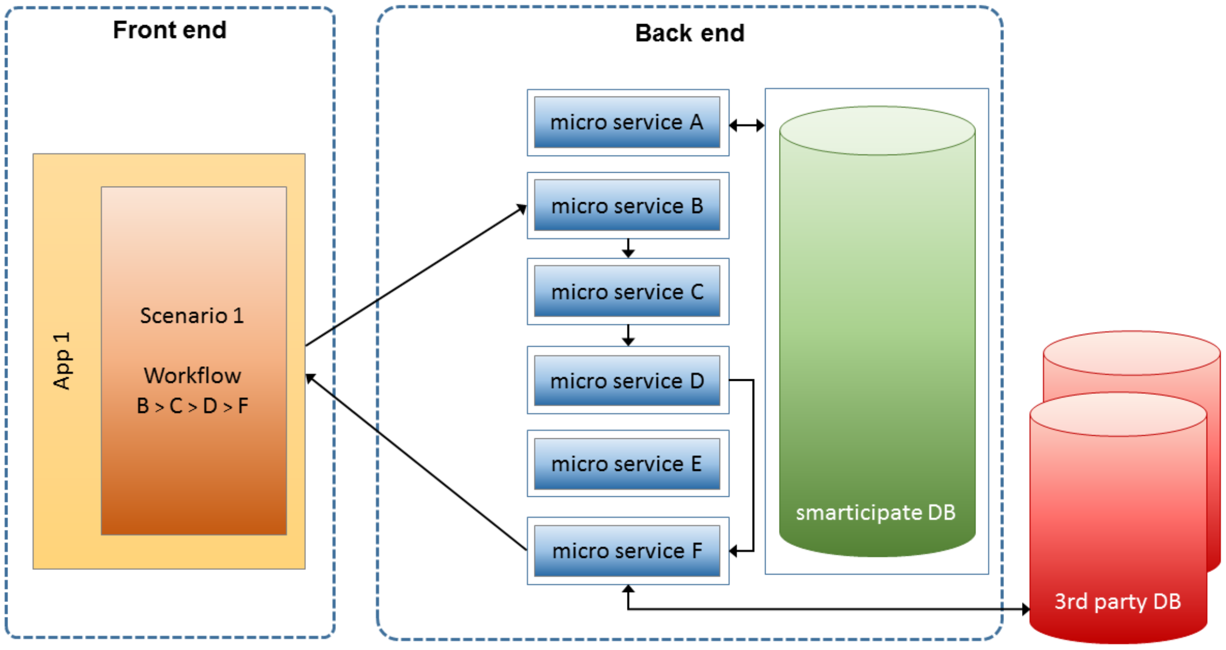


Figure 8: A request that prompts a chain execution of micro services (B, C, D and F) is executed with the services being aware of their successors and completing the workflow autonomously before feeding the result to the front end.

2.1.3 Logging

2.1.3.1 Logging standard

A common logging standard across all components is an essential requirement for effective error detection as well as detailed performance analysis. Logging of, for example, reading, writing and processing blocks allows developers and system analysts to create performance tests and compare run times for each of those blocks across different versions thus tracking the effectiveness of performance enhancement measures. In addition, such information allows the direct quantitative comparison between different environments and their effect on reading, writing and processing speeds (e.g. reading from different sources or processing in cloud or cluster environments). Figure 9 illustrates one of many possible test evaluations for a service or system component based on standardized logging of, in this case, reading, writing and processing blocks.

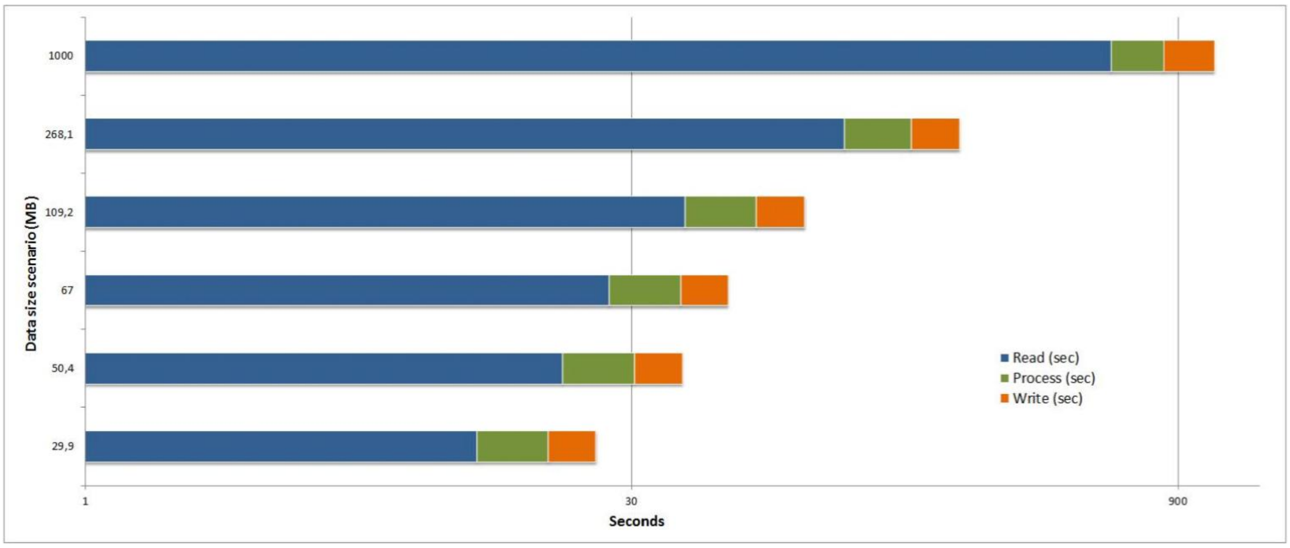


Figure 9: Reading, processing and writing block run times from a scalability test with increasing input data sizes on a logarithmic scale.

Developers of smarticipate components are therefore required to implement the following logging standard for all components developed for smarticipate:

<*{//}{DATETIMESTAMP;EPOCH;COMPONENT_ID;TYPE;BLOCK_OC;BLOCK_TYPE;DATA_SIZE_KB;MESSAGE}

Where

Table 2: Definition of logging standard

PLACEHOLDER	*MANDATORY*	*DESCRIPTION*	*EXAMPLE*
DATETIMESTAMP	***Yes***	[Text] Date/time stamp (UTC) with millisecond resolution in this format: ***dd.mm.yyyy_hh:mm:ss.zzz***	**09.12.2016_23:55:02.274**
EPOCH	***Yes***	[Long integer] Epoch timestamp in milliseconds	**1481327702274**
COMPONENT_ID	***Yes***	[Text or Integer] Unique ID of the smarticipate component (format to be agreed)	e.g. **MS004**
TYPE	***Yes***	[Text] Keyword **"INFO", "WARNING" or "ERROR"** indicating the nature of this message.	**INFO**
BLOCK_OC	***No***	[Text] Keyword **"open"**, **"close"** or for an opening or closing block. Leave blank if log neither opens nor closes a block.	**Open**
BLOCK_TYPE	***No***	[Text] Keyword **"read"**, **"process"** or **"write"** for a reading, processing or writing block. Leave blank if not applicable.	**write**
DATA_SIZE_KB	***Yes***	[List[Integer]] List of data sizes in kB handled by the service. Empty list if not applicable.	**[123,500079,452339]**
MESSAGE	***No***	[Text] Free text message	**Saving object to data base**

Examples

Opening of a reading block:

<*{//}{09.12.2016_23:55:02.274;1481327702274;MS004;INFO;open;read;[1024];start reading 1MB input file}

Closing the above reading block:

<*{//}{09.12.2016_23:55:04.555;1481327704555;MS004;INFO;close;read;[1024];input read: 1.0MB}

Random message during the execution of a block:

<*{//}{22.01.2017_13:00:37.260;1482823807260;MS033;INFO;;;[]; progress: 13/24 files}

The following logging rules apply:

- The entire run time of a component is the sum of its block run times. I.e. no runtime is unallocated to either a reading, processing or writing block.
- The first log line opens a block
- The last log line closes an open block
- Upon closing a block, another block is instantly opened in a new log line (except for the last log)

- An open block must be closed before another block can be opened. There must always only be one block open at any given time.
- Any log line that contains a value for **BLOCK_OC** must also hold a value for BLOCK_TYPE.

2.1.3.2 Log repository

All components will write their log messages to a server-side implementation of elasticSearch (https://www.elastic.co/products/elasticsearch) and logStash (https://www.elastic.co/products/logstash) to hold a common, queryable repository of all logs for analysis and error detection purposes.

2.2 Data specification

Open Geodata is at the core of the smarticipate platform. All smarticipate applications depend on the availability of freely accessible, standardized and quality assured geodata. This section will specify which data are of primary importance to the proposed scenarios covered by the smarticipate prototype and how this data is to be integrated and stored.

Please note, that this section complements the Data Management Plan (Deliverable D2.3) that will go beyond the described content.

2.2.1 Data storage

The smarticipate prototype will run a geodatabase as part of the backend that holds all available data as database objects. PostgreSQL (https://www.postgresql.org/) is a leading database management system (DBMS) that provides a wealth of data storage, manipulation and retrieval solutions. The PostGIS plugin (http://www.postgis.net/) adds extensive geo-functionality to a PostgreSQL database and allows the storage of many types of vector and raster formats. In the case of CityGML, there´s no ready counterpart data type in the database, however, the 3D CityGML DB extension (http://www.3dcitydb.org/3dcitydb/3dcitydbhomepage/) introduces support for CityGML (v 2.0 and 1.0) to PostgreSQL databases. Alternatively, GeoRocket (https://github.com/georocket/georocket) may be used to store 3D CityGML features. GeoRocket shall be evaluated and compared to 3DCityDB in terms of efficiency, stability and performance during the code camp in Vienna in February 2017. Either option is considered appropriate for the storage and retrieval of CityGML data.

The above combination of Open Source database solutions thus covers all the data storage needs of the project. Compatibility between the versions needs to be assured. At the time of writing the combination of PostgreSQL **9.6.1**, PostGIS **2.3** and 3DCityDB **3.3.1** were supported; GeoRocket **1.0** was pending release (scheduled for end of January).

2.2.2 Metadata standard

This section describes the metadata requirement for data to be integrated into the smarticipate platform. All data integrated into the smarticipate platform and/or linked to it through access and retrieval services (3rd party data repositories) must be accompanied with metadata that adhere to the standard defined below.

The harmonization will be executed according to the following applied ISO as well as relevant OGC standards. Standards are very useful when it comes to transnational projects in any type of business. The spatial data infrastructure in Europe is quite inhomogeneous (not standardised) and therefore limiting the transnational use of geodata. The INSPIRE Directive (2007/2/EC, Infrastructure for Spatial Information in Europe) has the aim to establish an infrastructure for spatial information in Europe that will help to make spatial or geographical information more accessible and interoperable. It addresses 34 data themes needed for environmental applications, with key components specified through technical implementing rules. As the process of implementing the rules of INSPIRE is currently at the beginning in Europe, INSPIRE compatible data are not available at this point of time. The consequence for the smarticipate project is a higher harmonisation effort as this would be the case after the successful implementation of INSPIRE.

INSPIRE defines its standards based on a number of ISO standards. ISO is a standards organization (by law), who's meetings take place on a country level. There is no membership by a company or research or local government organization - only at the national level. The OGC (Open Geospatial Consortium) can submit standards for processing and approval as ISO Standards and therefore in smarticipate we are only referring to ISO standards. Altogether there are currently nearly sixty ISO standards dealing with geoinformation (the so-called 191xx series) and this number is still growing. The ones relevant for smarticipate are ISO 19111 and ISO 19115.

ISO 19111 – Coordinate reference systems

ISO 19111 describes the conceptual schema and defines the description for a minimum data to two cases for which 1-, 2- and 3- dimensional coordinates reference system information shall be given. The first case is given by a coordinate reference system to which a set of coordinates is related. The second case consists of a coordinate operation (coordinate transformation, coordinate conversion, concatenated coordinate operation) to change coordinate values from one coordinate reference system to another.

There are no explicit accuracy numbers given in ISO 19111. We must consider that it has been developed for geographic information in general, but not for precise positioning. Spatial information may be referenced to the earth surface with an improving accuracy on the global scale for the future.

The spatial referencing is usually referred to selected points of the earth surface. Such point are, e.g., given by geodetic markers, stations performing permanent satellite observations, levelling benchmarks, or tide gauges. As soon as the marker coordinates are given, they provide a direct access to the realisation of the coordinate reference system.

ISO 19115:2003 – Metadata

ISO 19115:2003 defines the schema required for describing geographic information and services. It provides information about the identification, the extent, the quality, the spatial and temporal schema, spatial reference, and distribution of digital geographic data. ISO 19115:2003 is applicable to:

- the cataloguing of datasets, clearinghouse activities, and the full description of datasets;
- geographic datasets, dataset series, and individual geographic features and feature properties.

ISO 19115:2003 defines:

- mandatory and conditional metadata sections, metadata entities, and metadata elements;
- the minimum set of metadata required to serve the full range of metadata applications (data discovery, determining data fitness for use, data access, data transfer, and use of digital data);

- optional metadata elements - to allow for a more extensive standard description of geographic data, if required;
- a method for extending metadata to fit specialized needs.

2.2.3 Data integration

As far as data integration is concerned there are two distinct possibilities for that to happen.

The first is to couple the smarticipate platform with existing data repositories. This is done through micro services that provide access and retrieval capabilities to the repositories identified as relevant to the smarticipate platform.

The second is the integration of existing data that are not available through predefined portals but are made available to the project in the form of files or databases. In this case, an integration process chain is needed that assures the data are

- quality assured (accuracy, reliability, suitability - incl. metadata)
- converted (where necessary)
- imported into the project database

An assessment of the necessary tools and possible degree of automation of such pre-processing tasks can only be conducted once the available data repositories are described and a selection of desirable data sets is made. The latter task is pending a comprehensive listing and description of micro services to be developed. Figure 10 illustrates the preliminary pre-processing chain for manual to semi-automatic data integration.

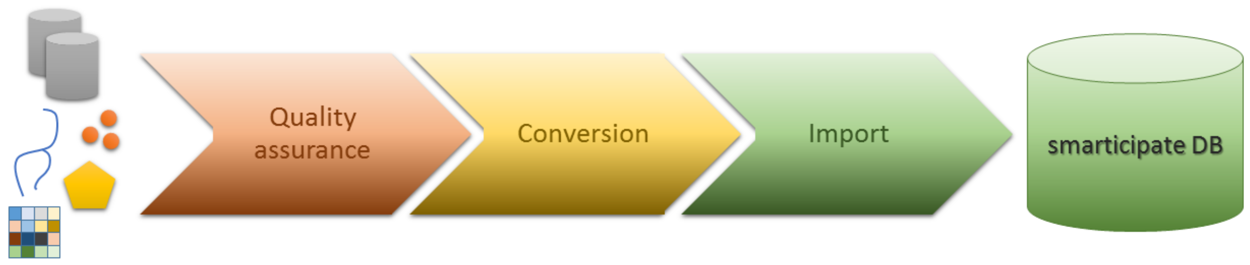


Figure 10: Preliminary data integration process chain for the integration of smarticipate data that is not otherwise available through an open third party data portal.

2.3 Data requirements

The following chapter summarises the data requirement specification activities by identifying data sets that are requested for the development of the smarticipate system. The availability of open data in cities is the basis for the development of the smarticipate system and will ensure successful implementation. In this respect, the use of city specific data enables end users to better contextually relate the smarticipate system to the local city specific scenarios. However, it is necessary to establish necessary data requirements often from user requirements definition or specification before the actual designing and implementation of the tools and applications begins. These data requirements enable end users to assess the extent to which the required data is available and can be provided.

In order to get better understanding of what data is needed to support the smarticipate system development and for which city, the following chapter gives an overview on these requested data sets. In general, these data requirements are derived from the use cases and system requirements specifications.

Explanation:

For smarticipate, high-resolution data is required in order to achieve appropriate results at local level. The geospatial data sets shall be provided as vector layers. The data should be provided in best case through the cities open data portal as WFS or WMS, in case the data are not openly available the provided standard format should be ESRI shape file or ESRI Geodatabase.

Geospatial data sets are required preferably as vector instead as raster layers. Statistical data must be provided at least for administrative sub-units in the city (smallest statistical unit) – like census districts, grid cells, blocks, to allow an investigation on share and the representativeness of the user sample.

2.3.1 Data requirements: Hamburg

Hamburg has a large amount of open data that is available through the cities open data portal (http://transparenz.hamburg.de/). The following table summarizes the overall data requirement specifications for smarticipate system based on the Hamburg scenario. In the case of Hamburg these data requirements specifications are directly related to the criteria and rules defined in *D3.2 Semantic data integration software and semantic representation concept*.

Id	Title	Description	Required data	Preferred format
HH-G	General data	General background data as base layer.	• 3D data • Administrative boundaries	• CityGML • WFS, WMS • Shapefile
HH-7	Infrastructure under ground	One of the most important aspects are infrastructure below the ground level, e.g. pipes for water or gas, power lines, phone and communication lines and so on. Data on these structures should be used to define areas to avoid planting as the tree roots may damage those infrastructures and also trees would have to be cut down in case a pipe needs to be repaired. A certain freely configurable distance to such infrastructure needs to be considered.	• Infrastructure below the ground level	• WFS, WMS • Shapefile
HH-8	Privately owned land	Privately owned land is excluded in all cases	• Ownership structure	• WFS, WMS • Shapefile

Id	Title	Description	Required data	Preferred format
HH-9	Land use and planned actions	Land that is already in use for buildings or streets is obviously not useable for planting trees. Also, planned actions should be considered if data is available. For example, if construction is planned for a street no new trees should be planted until the construction has been finished.	• Land use and land cover • Planning scheme (planned actions) • Binding land use plan • Imperviousness / soil sealing	• WFS, WMS • Shapefile
HH-10	Species is determined by neighbourhood of species	If for example an alley made up of all the same species of trees is given, a new tree should be of the same species, if the tree is reasonably close to the alley.	• Tree map including species (current and planned) • Tree costs • Location of parks	• WFS, WMS • Shapefile
HH-11	Species can be changed by definition	In contrast to the rule given above sometimes a tree species doesn't work out as desired on a certain location. Also, a possible climate change might influence the selection of trees to be planted. A rule should be implemented that overrides the rule of keeping the same species with a defined other species.	• CO2 emissions • CO2 calculations (to calculate reduction by species)	• WFS, WMS • REST Service → for real time information?
HH-12	Distance to street lighting	Trees grow and possibly will mask street lights nearby. A minimum distance should be kept from such positions. Positions of street lights need to be given.	• Above ground infrastructure ○ Location of street lights ○ Power supply lines	• WFS, WMS • Shapefile
HH-13	Distance to other trees	A certain distance to other trees is needed to avoid competition of both trees, for example sycamore trees need a distance of at least 8 meters, around 15 meters would be best.	• Information on given distances (based on city laws)	• WFS, WMS
HH-14	Distance to traffic signs or traffic lights	Trees grow and possibly will mask traffic lights nearby. A minimum distance should be kept from such positions.	• Above ground infrastructure ○ Location of traffic signs or traffic lights	• WFS, WMS • Shapefile
HH-15	Flooding areas	Areas which can be flooded should be avoided in general or a species that can cope with these needs to be selected.	• Flooding areas	• WFS, WMS • Shapefile
HH-16	Condition of soil	Basically, every ground close to road works is denaturised and	• Soil conditions	• WFS, WMS • Shapefile

Id	Title	Description	Required data	Preferred format
		needs to be refurbished. Though the surroundings of the potential tree position should be free of poisonous substances or demolition materials.		

2.3.2 London

Also London has a large amount of open data available which is a good basis for the system development (https://data.gov.uk/data/search?q=). Nevertheless, as no specific criteria and rules for the development are defined yet, the data requirement specifications are still on going in close cooperation with the developers. As soon as the definition of criteria and rules (as in Hamburg) is finalised, the following table will be updated.

Titel	Requested Format	Importance for use case scenario
3D model	• CityGML • VRML, X3D, 3ds, MAX (alternative)	If available, not essential
Ownership structure	• WFS, WMS • Shapefile	essential
Land use plan (Zoning Plan)	• WFS, WMS • Shapefile	essential
Land Cover (urban green structures, sealed/non-sealed)	• WFS, WMS • Shapefile	essential
Planning scheme (planned actions) Binding land use plan	• WFS, WMS • Shapefile	essential
Zoning information (allowed building types and height zones, areas classification based on commercial, housing, infrastructure, industrial and building restriction areas)	• WFS, WMS • Shapefile	essential
Heritage restrictions/guidelines	• WFS, WMS • Shapefile	essential
Administrative boundaries (e.g. districts) and postal codes	• WFS, WMS • Shapefile	essential
Population data (age, place of residence, place of work, education, …)	• WFS, WMS • Shapefile	essential
Commuting data	• WFS, WMS • Shapefile	low (may not be needed for test case)
Information about existing public transportation	• WFS, WMS • Shapefile	essential
Existing car parking areas, car sharing points, bicycle sharing points	• WFS, WMS • Shapefile	low

	• Tables	
Air quality data	• WFS, WMS • Shapefile	essential
Real estate cadastre	• WFS, WMS • Shapefile	essential
Building cadastre (including information on height, building material, usage, etc.)	• WFS, WMS • Shapefile	essential
Tree cadastre	• WFS, WMS • Shapefile	low
Guidelines for conversion	• tables	essential
Project guidelines	• tables	low
Value of property (market rates and social housing)	• tables	essential
Flood risk map	• WFS, WMS • Shapefile	low

2.3.3 Rome

Rome has currently only a medium amount of open data available (http://dati.comune.roma.it/). Similar to London, also in Rome no specific criteria and rules exist yet, that allows a more specific data requirement specification. Thus, the table needs an update as soon as the process of defining criteria and rules is finalised.

Titel	**Requested Format**	**Importance for use case scenario**
3D model	• CityGML • VRML, X3D, 3ds, MAX (alternative)	If available, not essential
Ownership structure	• WFS, WMS • Shapefile	essential
Land use plan (Zoning Plan)	• WFS, WMS • Shapefile	essential
Land Cover (urban green structures, sealed/non-sealed)	• WFS, WMS • Shapefile	essential
Planning scheme (planned actions) Binding land use plan	• WFS, WMS • Shapefile	essential
Heritage restrictions/guidelines	• WFS, WMS • Shapefile	essential
Administrative boundaries (e.g. districts) and postal codes	• WFS, WMS • Shapefile	essential

Population data (age, place of residence, place of work, education, …)	• WFS, WMS • Shapefile • Tables	essential
Commuting data	• WFS, WMS • Shapefile • Tables	low (may not be needed for test case)
Information about existing public transportation	• WFS, WMS • Shapefile	essential
Air quality data	• WFS, WMS • Shapefile	essential
Real estate cadastre	• WFS, WMS • Shapefile	essential
Building cadastre (including information on height, building material, usage, etc.)	• WFS, WMS • Shapefile	essential
Guidelines for conversion	• tables	essential
Project guidelines	• tables	low
Value of property (market rates and social housing)	• tables	essential

3 Deployment

3.1 Continuous integration (CI) / continuous Deployment (CD)

This section introduces the concept of Continuous Integration (CI) and Continuous Deployment (CD) proposed for smarticipate. Continuous Integration describes a process, where changes to the software (i.e. its code) are not merged at predetermined intervals but rather instantaneously as soon as changes become available. These changes are reflected in the master trunk. Similarly, Continuous Deployment refer to the process of continuously releasing new snippets to the live system (or building installations) whenever these new snippets have passed the testing stages. This method contrasts the previous regime of deployment in long release cycles. In essence, the combination of CI and CD ensure that ideas for new features or bug fixes run quickly through the cycle of coding, integrating, testing and release. The deployment process takes place in near real time resulting in releases shortly after an updated version of a component has been submitted by a developer. New uploads to a software repository instantly trigger a sequence of tests and, upon their successful completion, the deployment of that software component to the operational system. Continuous Integration and Continuous Deployment depend on a high degree of automation. It should start immediately when a developer checks code into the code repository. Since this build will be used directly for a software release, tests of the software are essential in order to avoid the integration of broken components. Such tests must ensure that the software not only runs but also behaves as expected. Upon passing the preconfigured tests, the software package is automatically deployed to the production environment. This concept has the advantage, that the used software packages will always be the most current version and have the newest bug fixes and features. The software will not be published to the platform without any tests, so a broken version of a service can be identified and intercepted before being used in the operational system. As an additional advantage, the developers need not be familiar with the actual deployment process to the platform. With this approach, the deployment process can support different architectures and infrastructures.

3.2 Component testing

In order to detect broken versions of a service, the automated tests are essential to the automatic deployment process. Without tests, there is a chance to use non-working services or services which will produce false results. The software testing is done in three stages.

First stage is the developer himself, who will perform continuous tests during the development of the software. This is known as white-box testing, since the developer knows the internal structure of his or her software. Usually this is carried out by unit testing, where each unit of the source code is tested individually. For this approach the developer should write test cases which are executed during the build time of the software packages. When a test case fails, the build is considered faulty and should not be deployed to the platform. Since the build will be created on the developer’s computer the smarticipate deployment system will not contain facilities for unit tests. In conclusion, this is in the area of responsibility of the individual developer and considered a pre-integration task.

The second stage of testing will be testing by a quality manager. In this way, a second person has to have a look at the software package before it can be deployed to the platform and used in workflows. This should at least include running the services on an example dataset.

The third stage of testing is the deployment service itself. It will use some automated tests, which must complete successfully, before the services are considered ready for deployment. These tests will run automatically, when a new release of a service is submitted to the software repository for the smarticipate platform. This will usually happen at the end of a sprint when new or updated components are released into the testing cycle external to the development. These tests will be run using GitLab (**Error! Reference source not found.**). For details on the development procedure and contents of SCRUM sprints please refer to D3.4, section 2.1. Upon successful completion of an integration test, the software package version is considered "ready for deployment". I.e. the software component can be used in the operational system. The deployment process itself will then be based on Docker Compose which specifies the setting and communication of a multitude of Docker containers.

3.2.1 GitLab

GitLab (https://about.gitlab.com/) is an online Git repository manager with a wiki, source control (versioning), coding interface, issue tracking, Continuous Integration and Continuous Deployment features (Figure 11).

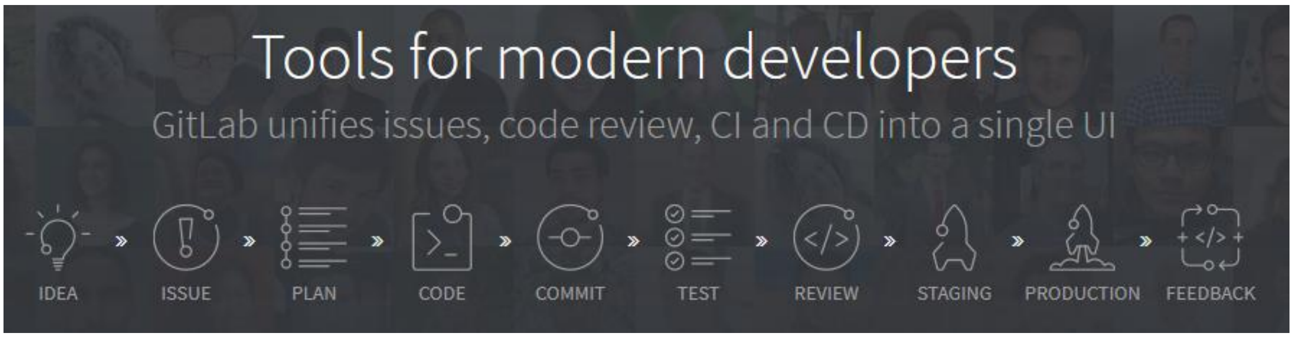

Tools for modern developers
GitLab unifies issues, code review, CI and CD into a single UI
IDEA
ISSUE
PLAN
CODE
COMMIT
TEST
REVIEW
STAGING
PRODUCTION
FEEDBACK

Figure 11: GitLab is a comprehensive solution for modern software development from start to finish.

It facilitates the management of git repositories on a centralized server. The free Community Edition has been set up for the smarticipate project to support the entire process chain described above. The comprehensive product suite of GitLab means that no external tools need to be used to complete the deployment pipeline, including automated testing with the CI feature.