



REPAIR

REsource Management in Peri-urban AREas: Going Beyond Urban Metabolism

D2.6 Technical Documentation

Version 2.0

Authors: Christoph Franke (GGR), Gustavo Arciniegas (Geo-Col), Rusné Sileryte (TUD), Max Bohnet (GGR), Jens-Martin Gutsche (GGR), Alexander Wandl (TUD), Stefanie Gutsche (GGR)

Grant Agreement No.:	688920
Programme call:	H2020-WASTE-2015-two-stage
Type of action:	RIA – Research & Innovation Action
Project Start Date:	01-09-2016
Duration:	48 months
Deliverable Lead Beneficiary:	GGR
Dissemination Level:	PU
Contact of responsible author:	repair@bk-tudelft.nl

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 688920.

Disclaimer:

This document reflects only the author's view. The Commission is not responsible for any use that may be made of the information it contains.

Dissemination level:

- PU = Public
- CO = Confidential, only for members of the consortium (including the Commission Services)

Change control

VERSION	DATE	AUTHOR	ORGANISATION	DESCRIPTION / COMMENTS
1.0	01-02-2020	Ch. Franke	GGR	FIRST DRAFT VERSION
1.1	01-09-2020	A. Wandl R. Sileryte B. Dukai T. Commandeur	TUD	Additions to all chapters through out the document
1.2	05-09-2020	Gustavo Arciniegas	Geo-Col	Additions to all chapters through out the document
2.0	01-10-2020	S.Gutsche	GGR	FINAL VERSION

Acronyms and Abbreviations

AJAX	Asynchronous JavaScript and XML
AoP	Area of Protection
API	application programming interface
ASMFA	Activity-Based Spatial Material Flow Analysis
CE	Circular Economy
CSRF	Cross-site request forgery
CSS	Cascading Style Sheets
DMP	Data Management Plan
EC	European Commission
EIS	Eco-Innovate Solution
EU	European Union
GDSE	Geo-design Decision Support Environment
GPL	General Public License
H2020	Horizon 2020
HTTP	Hypertext Transfer Protocol
JS	JavaScript
Lat	Latitude
LCA	Life Cycle Analysis
Lon	Longitude
MVC	Model-View-Controller
Nace	Statistical Classification of Economic Activities
NUTS	Nomenclature of Territorial Units for Statistics
ORBIS	Company Database
OSF	Open Science Foundation
PULL	Peri-Urban Living Labs
REST	Representational state transfer
SHP	Shapefile
SLD	Styled Layer Descriptor
UML	Unified Modeling Language
URL	Uniform Resource Locator
WFS	Web Feature Service
WMS	Web Mapping Service
WP	Work Package

Table of Contents

Change control	3
Acronyms and Abbreviations	4
Table of Contents.....	5
List of Figures	7
List of Tables	9
Publishable Summary	10
1 Introduction.....	11
1.1 Overview of the technical components of the GDSE	12
1.2 Hardware requirements.....	13
1.2.1 Hardware requirements for the Server (Backend)	13
1.2.2 Touch Table for the Frontend	13
1.2.3 Using the GDSE in Online-Workshops	14
2 Installation.....	17
2.1 Installation of the GDSE-Server	17
2.2 Installation of a development-environment for debugging	21
2.3 Testing, Branch Policy and Continuous Integration	22
3 Frontend and Backend Modules	24
3.1 Server backend	24
3.2 Web frontend	41
3.3 Internationalization	52
4 Data Management	53
4.1 Overview of the required data and user input	53
4.2 Material Flow Data preparation and Data Entry	65
4.3 Exporting Data	69
4.4 Integrating Geodata and Maps via WMS/WFS	72
4.5 Open Data Policy and Restrictions: User Management and Access Rights	80
5 Outlook of possible further development	82
Appendix	85
References	115

List of Figures

Figure 1: Technical components of the GDSE	12
Figure 2: Code sample - example of a docker-compose.py file with compose file version 3.1	20
Figure 3: Code sample - example of a .env file	20
Figure 4: Code coverage of 86% visualized in a circular diagram (at 12.08.2020)	23
Figure 5: Architecture of the Django framework (https://djangobook.com/mdj2-django-structure/).....	24
Figure 6: Code sample - example model "Activity" .\repair\apps\asmfa\models\nodes.py	25
Figure 7: Code sample - example serializer "ActivitySerializer" of the model "Activity"	25
Figure 8: Code sample - example view "ActivityViewSet" on the model "Activity"	26
Figure 9: UML of flow data	28
Figure 10: User access to case studies	29
Figure 11: UML-diagram of strategy/solution classes	29
Figure 12: Set up page of the permissions of the group "WorkshopParticipant" in the Django administration site	32
Figure 13: Screenshot of the HTML list-view on available casestudies https://gdse.h2020repair.bk.tudelft.nl/api/casestudies/	33
Figure 14: Screenshot of the REPAiR API Documentation https://gdse.h2020repair.bk.tudelft.nl/api/docs/	34
Figure 15: Example graph for material flows between actors.....	38
Figure 16: Code sample - use of collections - creation of a stakeholder category .\repair\js\views\study-area\stakeholders.js	42
Figure 17: Code sample - excerpt of routing config in .\repair\js\app-config.js.....	43
Figure 18: Code sample - embedded script of an underscore template inside the django template .\repair\templates\conclusions\workshop.html	44
Figure 19: Code sample - add a row showing a conclusion by rendering the template from code sample in Figure 18 \repair\js\views\conclusions\conclusions.js	44
Figure 20: Two rendered conclusion rows in the GDSE as a result of code sample 9 https://gdse.h2020repair.bk.tudelft.nl/conclusions/	45
Figure 21: Layout of same page at different resolutions. left: laptop with HiDPI screen (1440x900), right: iPad (768x1024)	46
Figure 22: Entry point to the JS scripts on a page	47
Figure 23: Visualization of flows with OSM background map (Jochim 2018)	50
Figure 24: Visualization of flows in the GDSE with overlay controls.....	50
Figure 25: Sankey diagram with overlay controls.....	51
Figure 26: Muuri container, draggable and movable by touch in the GDSE "Ranking Objectives"	52

Figure 27: An example of a successful table upload (green) and an error (red).	66
Figure 28: An example of a material hierarchy displayed in a tree-like structure	68
Figure 29: Red square indicates the button that allows downloading data which is visible in the displayed Sankey diagram.....	70
Figure 30: Adding a map layer to a category from the GDSE GeoServer.....	72
Figure 31: Adding an external WMS or WFS layer service in the Study Area step.....	79
Figure 32: Order of upload for different data sets.....	85
Figure 33: Relationship between all tables.....	89
Figure 34: Example of the Administrative Units for the Pecs case study.....	92
Figure 35: Example of an actor in the GDSE interface that can be found under “Data Entry” → “Edit Actors”.....	97
Figure 36: Example of the “Edit Materials” tab in the GDSE “Data Entry” interface	100
Figure 37: Data Sources.....	105

List of Tables

Table 1 Online GDSE workshop generic format.....	15
Table 2: Description of classes to be set up in preparation of workshops	30
Table 3: Description of classes holding inputs of workshop participants	30
Table 4: HTTP methods, the actions performed on request and the permissions required to execute the corresponding action	31
Table 5 parameters for filtering and aggregation flows.....	35
Table 6: Annotations to mark strings to be translated and the imports required to use the annotations in the different programming/markup languages used in the project.....	52
Table 7: Overview of the required data and user input	54
Table 8: The aggregation level of the data depends on the display level.....	71
Table 9: User Management and Access Rights	80
Table 10: Template of “Area Levels”	90
Table 11: Template of “Areas”	91
Table 12: Template of “Activity Groups”	93
Table 13: Template of “Activities”	93
Table 14: Template of “Actors”	95
Table 15: Example of household type actors	96
Table 16: Template of “Geolocated Actors”	98
Table 17: Example of the Materials hierarchy and its corresponding table for the GDSE..	99
Table 18: Template of the “Composition” table	101
Table 19: Dataset Administrative Units	108
Table 20: Dataset Materials.....	108
Table 21: Dataset Activity Groups	109
Table 22: Dataset Activity	110
Table 23: Dataset Actors.....	110
Table 24: Dataset Geolocated Actors	111
Table 25: Dataset Households	111
Table 26: Dataset Filtered actors.....	111
Table 27: Dataset Waste/ Product Composition	112
Table 28: Dataset Flows	113
Table 29: Dataset Stocks.....	113
Table 30: Dataset NACE-EWC correspondence table.....	114
Table 31: NACE-CPA correspondence table	114

Publishable Summary

The deliverable contains the technical documentation of the completed Geodesign Decision Support Environment (GDSE). The GDSE is the core digital support tool of the REPAiR project's approach and methodology. It is a web-based open source tool that adapts the geo design framework for the purpose of spatial diagnosis and creation of territorial and systemic eco-innovative strategies towards a Circular Economy.

The technical documentation consists of an introductory overview on the technical components of the GDSE, its licence and its hardware requirements, followed by step-by-step explanation of the installation process. After that, the documentation goes into the technical details of the frontend and the backend of the GDSE, explaining their structures, designs and functionalities as well as the components and libraries used for the programming. It then focuses on the data management, explaining the existing options for uploading, editing and exporting data into and from the GDSE. The documentation ends with an outlook on possible further developments and a list of follow-up repositories of other projects already using and modifying the GDSE.

1 Introduction

As described in previous deliverables, the Geodesign Decision Support Environment (GDSE) is the core digital support tool of the REPAiR project's approach and methodology. It is a web-based open source tool that adapts the geo design framework for the purpose of spatial diagnosis and creation of territorial and systemic eco-innovative strategies towards a Circular Economy (CE) (Arciniegas et al., 2019a).

While the previous deliverables focused on the concept and methodology (REPAiR, 2016; REPAiR, 2017a; REPAiR, 2017b) as well as the content and usage of the GDSE (REPAiR, 2019a; REPAiR, 2020), the focus of this deliverable is the technical documentation of the completed GDSE.

Therefore, the documentation in this deliverable mainly addresses an audience more or less familiar with database and web application technologies thinking about using the GDSE for their own research, consulting, development or planning purposes. For these readers, it aims to answer the two following questions:

1. How do I set up my own GDSE version to be used in my own research, consulting, development or planning?
2. How can I technically modify the GDSE so that it fits even better the needs of my own application context?

The technical documentation in this deliverable starts with an overview of the technical components of the GDSE, its licence and its hardware requirements, followed by a step-by-step explanation of the installation process. After that, the documentation goes into the technical details of both the frontend and the backend of the GDSE, explaining their structures, designs and functionalities as well as the components and libraries used for the programming. The next chapter focuses on the data management, explaining the existing options for uploading, editing and exporting data into and from the GDSE. The deliverable ends with an outlook on possible further developments and lists follow-up repositories of other projects already using and modifying the GDSE.

1.1 Overview of the technical components of the GDSE

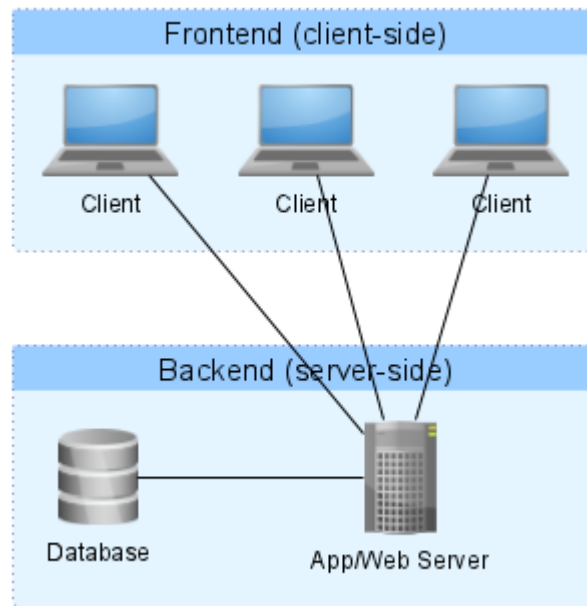


Figure 1: Technical components of the GDSE.

The GDSE is a Web-Application that runs on a web server. It can be accessed by multiple users (clients) in parallel via web browsers connected to the internet (see Figure 1).

The Frontend is written in HTML/JavaScript, the Backend is programmed with the Python framework Django 3.0. Next to the pure Python implementation of Django, several libraries must be installed:

- Gdal/GEOS: The GDAL/GEOS-Libraries are the standard OpenSource-Libraries for all geospatial operations
- Graph-tool: Graph-tool is a Python module used to model the impact of Eco-Innovative Solutions on upstream, downstream, and circular flows.
- Imagemagick/Ghostscript are required to process the uploaded pdfs
- SQLITE/Spatialite is a geospatial database used in the unittests
- Postgresql/PostGIS is used as a production database.
- CircleCI to manage the Continuous Integration during the Software Development
- May I use the GDSE in my own project? Licences, Property Rights etc.

The GDSE application is an open source project licensed with the GPLv2 (GNU General Public License, version 2). With this license, anybody is allowed to freely use, copy, further

develop and adapt the software for commercial or non-commercial purposes as long as the following conditions are met:

- the changed software in its entirety may only be given to third parties under the license conditions of GPLv2;
- according to sec. 1 GPLv2, a copy of the license text must be provided with the copy of the program;
- changed software files must contain a reference showing that changes were made, as well as the date of such changes (see sec. 2 GPLv2);
- sec. 1 GPL provides that a “suitable” copyright sign (in this case © 2020 REPAiR) must be affixed to each copyrighted item in an easily perceivable location;
- the corresponding source text in the form governed by sec. 3 GPLv2 must be made accessible well (e.g. on Github, Gitlab or similar platforms)
- it is impermissible to make the use of the software depend on additional obligations that are not listed in GPLv2, e.g. it is not allowed to protect the source code by incorporating it into a proprietary software.

The IP rights of the GDSE software at all times stay with the REPAiR consortium.

More information about the license and the license text can be found here:

<https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

1.2 Hardware requirements

1.2.1 Hardware requirements for the Server (Backend)

The GDSE requires a Linux-Server to run the Backend. For REPAiR, a linux-server with 32 GB RAM proved to be sufficient to run the GDSE in the workshops on various computers by various groups of participants. The PostgreSQL-Databases for REPAiR project measured < 1 GB.

1.2.2 Touch Table for the Frontend

The GDSE features interactive touch-enabled screens to facilitate on-location face-to face workshop communication in two ways: 1) between users and the GDSE frontend software, and 2) the dialogue between the GDSE users. Members of one small group (2 to 6 participants) use one touch table to interact with the GDSE frontend. Several touch tables are used as there are small groups created in one workshop. REPAiR uses touch tables with

a diagonal screen size of 28 inches that can be easily switched between horizontal (discussion) and vertical (presentation) mode. Larger touch tables can also be used as long as the computer linked meets the hardware requirements.

1.2.3 Using the GDSE in Online-Workshops

The GDSE was developed as a web application that can be accessed on any browser provided there is an internet connection and intended users have the right credentials to access it. This allows PULL workshops to be held both remotely or on location. Online GDSE-centred PULL workshops are hosted as webinars using video/web conferencing platforms that allow to split the meeting in separate sessions (also known as breakout rooms), allow screen sharing, and feature a chat box for interaction between participants and the PULL hosting team members. In this way, the workshop participants are divided into small groups, which can interact via chat messages with each other and the PULL hosting team.

REPAiR organizes online GDSE workshops using the Zoom video conference platform's breakout rooms to host all participants and manage the small groups. The work of each small group is facilitated by a PULL hosting team member in each breakout room, who acts as the 'pen holder' in the GDSE. The Zoom chat box is used for sharing hyperlinks with users, for example to online pre- and post-workshop surveys (prepared using Google Forms), instruction videos (prepared using YouTube), and further instructions in the form of text. Through screen sharing, participants can see in real time their work and progress.

The general goal of an online PULL workshop is to have each small group co-develop one eco-innovative strategy for a particular PULL-specific key flow, addressing the objectives defined in early stages of the PULL process, and utilizing the Eco-Innovative Solutions (EIS) previously defined in the catalogue (see the REPAiR catalogues of solutions and strategies for all six cases in deliverables 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8). Each small group uses the GDSE to follow the process of

- ranking objectives per (small) decision-maker group;
- setting resource flow targets the group wants to achieve;
- co-developing one strategy per small group;
- assessing the changes in terms of flows the strategy achieves in relation to the targets set.

The workshop follows the following generic format:

Table 1 Online GDSE workshop generic format

Activity	Hosting team prepares
Welcome and surveys	Pre-workshop online survey [LINK provided in chat box] Consent forms [LINK provided in chat box]
Split into Breakout rooms (small groups)	Breakout rooms
Welcome presentation Intro and aim of the workshop	Guiding online presentation about the PULL case [URL to slideshow], using 'share screen' function
Introduce Subgroups	Explain and define 'pen holder' for subgroups A quick presentation on parts of GDSE
Work in the main room: Study area	On GDSE show: a. A quick presentation on parts of GDSE b. Basic layers of infrastructure and demography in categories (demo of maps) c. List of stakeholders involved d. Definition of the key flow
Status Quo	a. Flows - a maximum of six flow views. Show users how to aggregate on different levels and how to select features to be visualised on the map. b. Flow indicators prepared c. Wastescapes: layers of the wastescapes maps briefly explained Objectives – show the list of objectives
Sustainability assessment status quo	Results of the sustainability assessment of the case are presented in the GDSE, show different results of the AoP Introduce Subgroups 'Pen holder' for subgroups
Split into breakout rooms	Login in and have users test the functionality of tool Share screen video 'How to log in link' [URL also provided in chat box] Credentials for small groups
Send instructions video for the next step to the breakout rooms/ demonstrate the next step using a shared screen.	
Target Objective –objectives Video link targeting objective	Ask users to rank the objectives Discussion follows If there is a disagreement, have them vote.
Send instructions video to breakout rooms Setting Flow targets/ demonstrate the next step using a shared screen. Video link URL provided in chat box	Have them discuss which targets they want to set to which indicators and related objectives. There are three different indicators: <ul style="list-style-type: none"> waste produced by number of inhabitants waste produced by area waste going into incineration if there is disagreement among the stakeholders note it down in the notes tab.

<p>Send instructions video to breakout rooms:</p> <p>Strategy / demonstrate the next step using a shared screen.</p> <p>Video link URL provided in chat box</p>	<p>a. Solutions - show the list of solutions demonstrate the information for one and let them look at the others (use only tabs Description and CE-Diagrams)</p> <p>b. Define Strategy - show them the steps of defining a strategy and let them discuss which solutions they want to combine. (minimum 3 solutions max 5)</p> <p>c. After they finished, define the strategy press calculate</p> <p>Coffee break</p> <p>d. Investigate the modified flows;</p> <p>e. See target control -check how the strategy related to their targets;</p>
<p>Coffee break</p> <p>Back to the main room</p>	<p>GDSE step Conclusions logged on as data captain</p>
<p>Plenary discussion: comparison of strategies, feedback exchange</p>	<p>Conclusions</p> <p>a. Show a comparison of the objectives between the groups, ask groups for the arguments for their ranking;</p> <p>b. Show the comparison of the strategies groups present the argument for their strategy</p>
<p>Closing</p>	<p>Reflections</p> <p>Post-workshop Survey [LINK in chat box]</p>

In order to make the online sessions of the PULL meetings easier to handle, simple step-by-step videos have been produced for those steps in the GDSE that can be followed by a group without moderation. The aim of the videos is to have multiple small stakeholder groups working in parallel at the same time.

The following videos are available online:

1. How to log in. [link to video](https://www.youtube.com/watch?v=EwbobYRvwJI&feature=youtu.be)
(<https://www.youtube.com/watch?v=EwbobYRvwJI&feature=youtu.be>)
2. How to rank objectives. [link to video](https://www.youtube.com/watch?v=VvcUZBDd5Ww&feature=youtu.be)
(<https://www.youtube.com/watch?v=VvcUZBDd5Ww&feature=youtu.be>)
3. Setting Flow targets. [link to video](https://www.youtube.com/watch?v=zMuks-KFdFg&feature=youtu.be) (<https://www.youtube.com/watch?v=zMuks-KFdFg&feature=youtu.be>)
4. Defining a strategy. [link to video](https://www.youtube.com/watch?v=zMuks-KFdFg&feature=youtu.be) (<https://www.youtube.com/watch?v=zMuks-KFdFg&feature=youtu.be>)

2 Installation

2.1 Installation of the GDSE-Server

Requirements

Linux is the recommended Operating System to run the server in. Some libraries are difficult to install under Windows, especially graph-tool, the library to build and analyse graphs.

The required libraries are the following:

- Python 3.6 including pip
- GDAL 2 and GEOS
- PostgreSQL libraries with PostGIS extension
- Imagemagick and ghostscript for PDF conversion
- git to pull the code from the REPAiR-repository
- gettext for internationalization
- node and yarn to manage the JavaScript-modules
- graph-tool and its dependencies pycairo and pygobject

Database

Theoretically, any database with a spatial extension supported by Django can be used to store and access the data required by the GDSE. Nevertheless, the REPAiR GDSE was developed to be used with PostgreSQL. Spatialite should work as well, but is not recommended to use productively due to problems with concurrent accesses.

If you want to use PostgreSQL as your backend, you have to create a blank database first. The database also has to support the POSTGIS extension. To protect the data and user inputs, you should set up regular backups of this database.

Run the Django server

To prepare and start the Django server, the following steps have to be executed in the directory you want to install the server to:

- Clone the repository

```
git clone https://github.com/MaxBo/REPAiR-Web.git
```


- Enter the directory that was created while cloning

```
cd REPAiR-Web
```

- Pull the latest version of REPAiR-Web from the master-branch in Github (not necessary directly after cloning)

```
git pull
```

- Install the requirements in python with pip

```
pip install -r requirements-dev.txt
```

- Install the JavaScript-Requirements with Yarn

```
yarn install
```

- Bundle the Java-Script-modules with webpack

```
node_modules/.bin/webpack-config repair/webpack.prod.config.js
```

- Collect the static Files used in the website (images, Logos etc.)

```
python manage.py collectstatic --noinput
```

- Migrate the Database to apply the latest changes in the Database scheme

```
python manage.py migrate
```

- Compile the messages (translation of the website into different languages)

```
python manage.py compilemessages
```

- Start the Django server locally on a specific port

```
python manage.py runserver localhost:<port>
```

Settings and environment variables

The configuration of Django is done with Python settings-files. Those files are located in the sub-directory *repair* of the installation directory. In there you will find the file *settings.py* with the basic settings. You should at least adapt the variable `ALLOWED_HOSTS` there. It contains the host/domain names that this Django site can serve (see <https://docs.djangoproject.com/en/3.1/topics/settings>). All other settings are based on the *settings.py* but define different database and debug settings.

The productive server of the REPAiR-GDSE is running with the settings defined in *settings_prod.py*. It is configured to use a PostgreSQL backend. If you want to use a similar setup, you can use this one as a template and set a different host and port. The credentials of the database are not written into the settings in plain text but passed by environment variables `DB_NAME`, `DB_USER` and `DB_PASS`.

Do not put any credentials into the settings files ever or otherwise anybody can see them on Github after pushing them. Use environment variables instead. That applies to the secret key as well. It should be overwritten by the environment variable `SECRET_KEY` in a productive setup.

After defining the settings, you have to inform Django which settings-file to use. You can do this either by passing the keyword-argument `--settings=<filename without extension>` to run the `manage.py` script or by setting the environment variable `DJANGO_SETTINGS_MODULE` (e.g. `DJANGO_SETTINGS_MODULE=repair.settings_prod`).

Under a Linux environment you additionally have to tell Django where the GDAL installation is located with the environment variable `GDAL_DATA`.

Next to Django, webpack needs to be configured. Webpack is used in this project to bundle the JavaScript modules. The JavaScript-configuration-files define amongst other things the paths to the scripts, the entry points and the debug-settings. Different configurations are provided in the directory `.\repair` including settings for the productive and staging servers.

Using Docker

The installation of the required libraries is not straightforward due to the complex dependencies of these libraries. Therefore, using a Docker Container to install the libraries in an isolated environment is highly recommended.

In the scope of the REPAiR project a container able to run the GDSE was created. It is built on Linux Debian 9 (Stretch) as a base image. Next to the libraries mentioned in the section [Requirements](#), it contains support for spatialite databases and CircleCI. CircleCI is included because the same image is used for running the tests to support a continuous development with CircleCI ([see Testing, Branch Policy and Continuous Integration](#)).

The docker image is available at

<https://hub.docker.com/repository/docker/maxboh/docker-circleci-node-miniconda-gdal>
(tag `graph_tool_stretch`)

To pull the image, run

```
docker pull maxboh/docker-circleci-node-miniconda-gdal:graph_tool_stretch
```

The image is automatically built based on the Dockerfile available at

https://github.com/MaxBo/docker-circleci-node-miniconda-gdal/tree/graph_tool_stretch

If you use the docker container to run the GDSE, the installation of the libraries mentioned above is not required. Instead, just Docker has to be installed on the machine running the server.

To run the server within the pulled docker container you have to proceed the steps described under [Run the Django server](#) inside the container.

Alternatively, you might create a docker-compose.yml. The following docker-compose.yml automatically pulls the container and the repository, starts the server and keeps the server alive on restart. See Figure 2 below.

```
version: '3.1'

services:
  web:
    env_file: .env
    image: maxboh/docker-circleci-node-miniconda-gdal:graph_tool_stretch
    command: bash -c "cd /home/circleci/repairweb && echo $PWD && git pull &&
    git checkout stable && pip install -r requirements-dev.txt && yarn install &&
    node_modules/.bin/webpack --config repair/webpack.prod.config.js && python
    manage.py collectstatic --noinput && python manage.py migrate && python
    manage.py compilemessages && python manage.py runserver 0.0.0.0:8000"
    ports:
      - "${REPAIRPORT}:8000"

  volumes:
    - ./static:/home/circleci/repairweb/repair/public/static
    - ./media:/home/circleci/repairweb/repair/public/media

  restart: always
```

Figure 2: Code sample - example of a docker-compose.py file with compose file version 3.1

In the example above the productive branch is called “stable”. The environment variables are set in the .env-file. The .env-file has to contain all environment variables described in the section [Settings and environment variables](#) especially the variable DJANGO_SETTINGS_MODULE to tell Django which settings-file to use. Furthermore, the static and media directories are mapped to directories outside the container.

If you are using the docker-compose.yml as shown above, you also have to define the variable REPAIRPORT to set the port in the environment running the container. See Figure 3 below.

```
DJANGO_SETTINGS_MODULE=repair.settings_prod
DB_NAME=gdse
DB_USER=[...]
DB_PASS=[...]
SECRET_KEY=[...]
REPAIRPORT=8001
```

Figure 3: Code sample - example of a .env file

To start the server with docker-compose type the following inside the directory with the docker-compose.py file:

```
docker-compose up -d
```

To stop the server type

```
docker-compose down
```

To apply changes made to the branch the docker-compose file pulls from to the server, the easiest way is to stop and restart the server.

Finally, you have to expose the chosen port and grant access to the static and media directories. How this is done depends on the HTTP server software used on the server.

2.2 Installation of a development-environment for debugging

To set up a development environment, you first have to dissolve all dependencies listed under [Requirements](#). Under Windows, you may skip the installation of graph-tool because there are no graph-tools binaries available for windows and compiling graph-tools for windows is not supported by the time of writing. Be aware that you cannot trigger any calculations of strategies if you skipped the graph-tool installation. All other modules should work without it.

It is recommended to create a local PostgreSQL-database for debugging. Alternatively you may use a configuration for a spatialite-database (`.\repair\settings_dev.py`) or create a custom settings file to connect to the productive database. However, the latter is not recommended because all migrations in development will be applied to the productive database and might not work with the state of the productive branch. Regardless which configuration you choose, the variable `DEBUG` should be set to `True` in your settings. The way to set up the environment variables mentioned in the section [Settings and environment variables](#) depends on the IDE you are using to debug.

Before starting the server you have to take the same steps mentioned under [Run the Django server](#) except the webpack-bundling. It is recommended to run all python commands in an environment, either a virtualenv or a conda environment. To bundle the JavaScript files in a debug environment run the `server-dev.js` script provided in the root directory of the installation:

```
node server-dev.js
```

The script runs a server providing the bundled scripts locally on port 8001. It keeps track of changes to the JavaScript files and automatically rebundles them without the need to restart. To change the port, you have to manually edit the files `server-dev.js` and `<ints.dir>/repair/webpack.dev.config.js`.

Another option is to debug in a vagrant container. There is a detailed description of how to do this under <https://github.com/MaxBo/REPAiR-Web/blob/master/VAGRANT.md>.

2.3 Testing, Branch Policy and Continuous Integration

When you want to adapt the GDSE to your own needs, make a fork of the repository. Do this either on the Github repository webpage or with Git console commands. The following paragraphs describe the recommended way to develop as done in the REPAiR project.

There are two separate servers running the GDSE: the productive server at <https://gdse.h2020repair.bk.tudelft.nl> and a development server at <https://staging.h2020repair.bk.tudelft.nl>. The development server is in the state of the master branch of the Github repository. It runs in debug mode and is meant to test changes to the user interface and the interactions with the backend before going into production.

The productive server is linked to the branch “stable”. After changes are approved, the master should be merged into the stable branch. The productive server is running in a docker container executing all necessary pulls and migrations, so a restart of the container is sufficient to apply the changes. If your productive server does not contain automatic migrations etc. on startup, you have to rerun specific single steps mentioned under [Run the Django server](#) depending on the changes compared to the previous state.

The master branch itself is protected. This means that changes to the code cannot be pushed directly to the master branch but separate (feature) branches have to be used and merged into the master via pull requests. The repository is configured to run tests before a pull request is accepted. Therefore, it is connected to CircleCI as a Continuous Integration system. The configuration can be found in the file `.\circleci\config.yml`. The tests are located in the directories of the apps.

Next to the tests, CircleCI is configured to keep track of the amount of code covered by the tests. Ideally the code coverage is around 100% to ensure a stable runtime of the server. The code coverage of the REPAiR project can be seen in the figure below (figure 4).



Figure 4: Code coverage of 86% visualized in a circular diagram (at 12.08.2020)

In summary, the workflow to implement new features in the continuous integration environment involves the following steps:

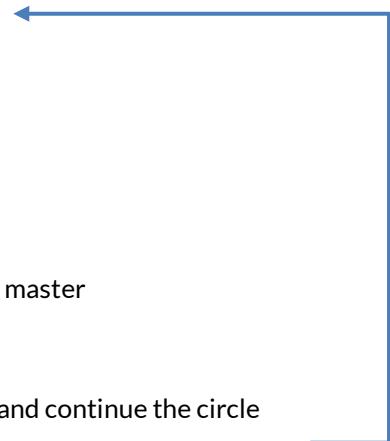
- 1) Create a new branch or feature branch.
- 2) Write a test.
- 3) Implement the code.
- 4) Commit and push to your branch.
- 5) Create a pull-request on Github.
- 6) Wait for CircleCi to run the tests.

If build was successful:

- a) Merge your branch into the master

else:

- b) Enter step 3 to fix the code and continue the circle



3 Frontend and Backend Modules

3.1 Server backend

The server backend of the GDSE is written in Python 3 and based on the Django web framework. Its main purpose is to listen to incoming requests and to manage, filter and provide the data to the frontend. It also serves the templates and static files required by the frontend to view the website.

Basic architecture

The architecture of the backend is determined by the architecture of the Django framework, which is similar to the Model-View-Controller pattern (see Figure XX). It does not strictly follow this pattern but tries to keep the data, the view logic and the business logic separate. There is no clear distinction in Django between View and Controller. Therefore, Django is sometimes referred to as a loosely coupled framework with a Model-View-Template pattern. <https://djangobook.com/mdj2-django-structure/>

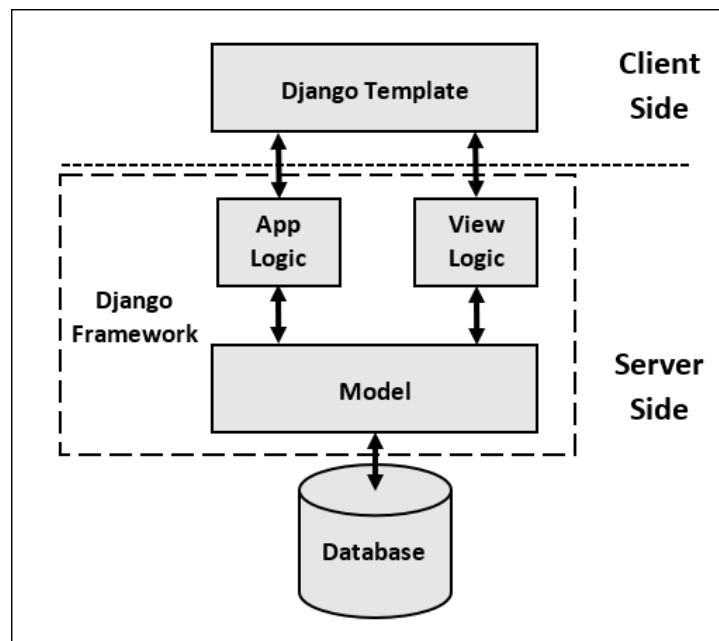


Figure 5: Architecture of the Django framework (<https://djangobook.com/mdj2-django-structure/>)

The Django models provide access to resources inside the database and contain the business logic, how the data can be created, stored and changed. A model represents a database table with the fields as attributes of the model. The code sample in Figure 6 below

shows an excerpt of a simple model representing the activities. Each activity has a name, a code (Nace-Code) and refers to an Activity-Group.

```
class Activity(Node):
    nace = models.CharField(max_length=255)
    name = models.CharField(max_length=255)
    activitygroup = models.ForeignKey(ActivityGroup,
                                     on_delete=PROTECT_CASCADE)
    [...]
```

Figure 6: Code sample - example model "Activity" .\repair\apps\asmfa\models\nodes.py

The serialization of resources for the purpose of transferring them to the client and the deserialization back into models is done by the Django REST framework. The framework also allows the implementation of a [RESTful API](#) to access the views on the resources. A serializer for the activity model is shown in the next code sample.

```
class ActivitySerializer(CreateWithUserInCasestudyMixin,
                        NestedHyperlinkedModelSerializer):
    parent_lookup_kwargs = {
        'casestudy_pk': 'activitygroup__keyflow__casestudy__id',
        'keyflow_pk': 'activitygroup__keyflow__id'
    }
    activitygroup = IDRelatedField()
    activitygroup_url = ActivityGroupField(
        view_name='activitygroup-detail',
        source='activitygroup',
        read_only=True)
    activitygroup_name = serializers.CharField(
        source='activitygroup.name', read_only=True)
    flow_count = serializers.IntegerField(read_only=True)
    [...]
```

Figure 7: Code sample - example serializer "ActivitySerializer" of the model "Activity"

Django views take requests from the clients and return appropriate responses. Depending on the purpose of the view the response can either be a web page or a serialized resource. Access to the views and resources is provided by URLs via HTTPS. The mapping of urls to the views is done in the file .\repair\urls.py. The code sample in Figure 8 shows a view that returns information on one or several activities as an answer to a GET-Request and handles the data to create a new activity sent by a POST-Request.

```
class ActivityViewSet(PostGetViewMixin, RevisionMixin,
                    CasestudyViewSetMixin, ModelPermissionViewSet):
    pagination_class = UnlitedResultsSetPagination
    add_perm = 'asmfa.add_activity'
    change_perm = 'asmfa.change_activity'
    delete_perm = 'asmfa.delete_activity'
    serializer_class = ActivitySerializer
    queryset = Activity.objects.order_by('id')
    serializers = {'list': ActivityListSerializer,
```



```
'create': ActivityCreateSerializer}  
  
def get_queryset(self):  
    [...]
```

Figure 8: Code sample - example view “ActivityViewSet” on the model “Activity”

Templates contain the actual design of the web page as HTML. Next to static content, they have a special syntax to generate dynamic content. The system used to fill the dynamic parts is the Django template language. The templates are put together and provided by specific views on request and are then rendered client-side (see [Web frontend](#)).

Project Structure

The code is organized in separate applications. Each app holds specific models, views, serializers and tests. The apps are located at `\repair\apps`.

The most basic GDSE models are inside the app “login”. That includes the models and views related to the user management and the case studies.

The most extensive app is the “asmfa” app. “asmfa” means “Activity-Based Spatial Material Flow Analysis”. It contains the models related to material flow data like actors, key flows and materials. It is also home of the [Graph Walker](#) algorithm to calculate the impact of strategies on existing flows and the [Flow Filters](#) and flow aggregation functions. It has no views on web pages.

The HTML web page views and the URLs leading to them are part of the apps that represent the steps of the GDSE decision-making process: “data-entry”, “study-area”, “status-quo”, “targets”, “changes” (step “Strategy”), and “conclusions”. These apps also contain models storing client-side user inputs like defined strategies and indicator settings.

The apps “publications”, “wmsresources” and “reversions” are modifications of existing 3rd party extensions with custom bug fixes and adaptations to the needs of the GDSE.

Data Structure

The flow data used for the AS-MFA is acquired on an actor level. Originally it was intended to store the data between activities and activity groups as well. To avoid redundancies and mismatching results on different observation levels, it was decided to store flows between actors only and aggregate the data to the activity or activity group level on demand (see

[Flow Filters](#)). Remnants of the data structure to store the flows on different levels can still be found throughout the code but do not serve any purpose anymore.

The flow data is initially stored with compositions of materials flowing from an origin to a destination actor. Compositions are either waste or products. They are composed of fractions of materials as visualized in the box named “old structure” in the UML (Unified Modelling Language) diagram in Figure 9. The advantage of this structure is that compositions are generalizable and can be reused in different scenarios. The concrete composition of materials is often not known but derived statistically and sampled anyway. This applies to the sampling of the composition of residual waste for example (REPAiR, 2019b).

During the project, this structure of flows, compositions and fractions turned out to be overly complicated. In particular, some of the required filter functions could hardly be realized due to the complex relations. A new structure was introduced merging the separate tables of compositions, fractions, stocks and flows into a single one - the *FractionFlow* (see Figure 9). Every flow of material is represented by a single flow instead of being bundled in flows of compositions. Stocks are no longer needed as a separate model but represented by the *FractionFlow* as well by having no destination.

Both structures still exist next to each other in the database at the moment. The old structure is still used to bulk upload data (see Annex 1) and in the data entry. The transformation of the flow data into the new *FractionFlow* format is triggered automatically on change or creation. A single flow from Actor to Actor (*Actor2Actor flow*) might be split into several *FractionFlows*, one for each fraction in the original composition of

a flow. The *FractionFlow* keeps the information about which *Actor2Actor* flow it was created from.

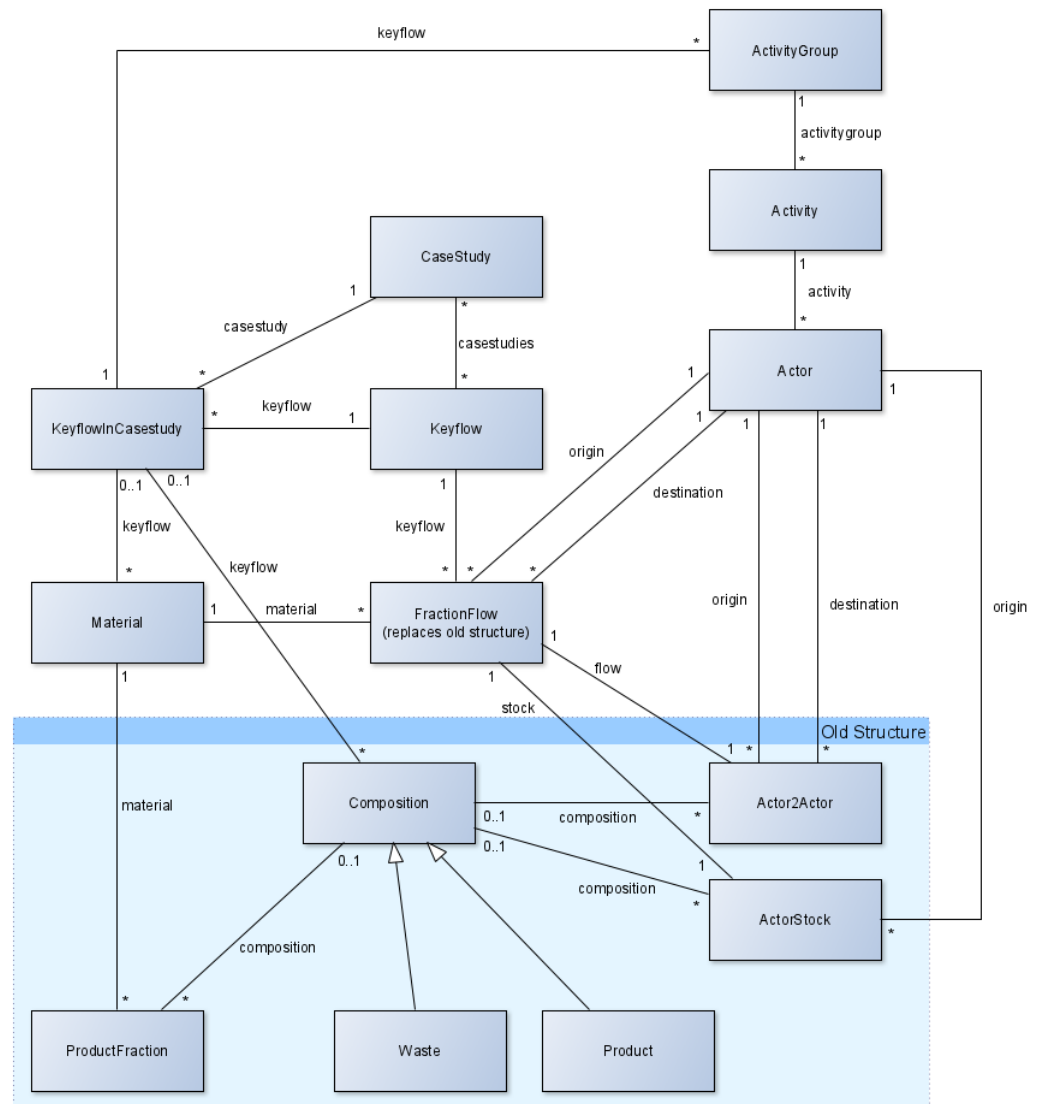
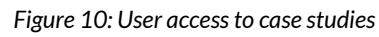


Figure 9: UML of flow data

The default Django user management is extended in the GDSE (see Figure 10). Every user has a Profile that defines which case studies can be accessed by the user and persists the user session attributes (see [Sessions](#)). Furthermore, the profile can be related to by multiple *UserInCasestudy* entries. The *UserInCasestudy* is used to assign strategies and to determine which users will be compared in the conclusions.



logic: how schemes are translated

The UML class diagram illustrates the relationships between various entities, categorized into two main groups: **user inputs (workshop)** and **setup (workshop preparation)**.

Entities:

- UserInCasestudy** (1 instance)
- KeyflowInCasestudy** (1 instance)
- Strategy** (* instances)
- Stakeholder** (1 instance)
- SolutionCategory** (1 instance)
- SolutionInStrategy** (1 instance)
- Solution** (1 instance)
- SolutionPart** (* instances)
- ImplementationQuantity** (* instances)
- ImplementationQuestion** (1 instance)
- FlowReference** (* instances)
- Activity** (1 instance)
- Process** (1 instance)
- Material** (1 instance)
- ImplementationArea** (* instances)
- PossibleImplementationArea** (1 instance)
- AffectedFlow** (* instances)

Relationships:

- UserInCasestudy** (1) is associated with **Strategy** (*) via the role **user**.
- KeyflowInCasestudy** (1) is associated with **Strategy** (*) via the role **keyflow**.
- KeyflowInCasestudy** (1) is associated with **SolutionCategory** (1) via the role **keyflow**.
- Strategy** (*) is associated with **Stakeholder** (1) via the role **coordinating_stakeholder**.
- Strategy** (1) is associated with **SolutionInStrategy** (*) via the role **strategy**.
- SolutionInStrategy** (1) is associated with **ImplementationQuantity** (*) via the role **implementation**.
- SolutionInStrategy** (*) is associated with **Solution** (1) via the role **solution**.
- Solution** (1) is associated with **ImplementationQuestion** (1) via the role **question**.
- Solution** (1) is associated with **SolutionPart** (*) via the role **solution**.
- SolutionPart** (*) is associated with **FlowReference** (*) via the role **flow_reference**.
- SolutionPart** (*) is associated with **FlowReference** (*) via the role **flow_changes**.
- FlowReference** (*) is associated with **Activity** (1) via the role **origin_activity**.
- FlowReference** (*) is associated with **Activity** (1) via the role **destination_activity**.
- FlowReference** (*) is associated with **Process** (1) via the role **process**.
- FlowReference** (*) is associated with **Material** (1) via the role **material**.
- FlowReference** (*) is associated with **AffectedFlow** (*) via the role **affected_flows**.
- AffectedFlow** (*) is associated with **Process** (1) via the role **process**.
- AffectedFlow** (*) is associated with **Material** (1) via the role **material**.
- AffectedFlow** (*) is associated with **Material** (1) via the role **origin_activity**.
- AffectedFlow** (*) is associated with **Material** (1) via the role **destination_activity**.
- ImplementationQuantity** (*) is associated with **ImplementationArea** (*) via the role **question**.
- ImplementationArea** (*) is associated with **PossibleImplementationArea** (1) via the role **possible_implementation_area**.
- PossibleImplementationArea** (1) is associated with **ImplementationArea** (*) via the role **implementation**.
- PossibleImplementationArea** (1) is associated with **ImplementationArea** (*) via the role **implementation**.

Figure 11: UML-diagram of strategy/solution classes

Table 2: Description of classes to be set up in preparation of workshops

CLASS	DESCRIPTION
SolutionCategory	<ul style="list-style-type: none"> ● categorizes solutions for clarity ● key flow specific
Solution	<ul style="list-style-type: none"> ● defines, how the solution changes flows ● instructions for graph walker algorithm ● consists of multiple solution parts
SolutionPart	<ul style="list-style-type: none"> ● single step in a solution ● detailed instruction how to change a set of flow ● <i>flow_reference</i> marks flows, that are directly changed by the SolutionPart ("before") ● <i>flow_changes</i> describes, how marked flows change their attributes ("after") ● <i>affected_flows</i> mark further flows to be impacted indirectly by changes
FlowReference	<ul style="list-style-type: none"> ● description to filter flows ("before") respectively instruction to change attributes of flows ("after") ● Attributes (all optional) <ul style="list-style-type: none"> ○ <i>origin_activity</i> (changeable) - activity of the origin actors ○ <i>destination_activity</i> (changeable) - activity of the destination actors ○ <i>material</i>, <i>hazard</i>, <i>waste</i> or <i>product</i>, <i>process</i> (changeable) ○ <i>origin_area</i> (only for description) - spatial area origin actors are located in ○ <i>destination_area</i> (only for description) - spatial area destination actors are located in
AffectedFlow	<ol style="list-style-type: none"> a. description of flows, similar to FlowReference b. used to mark flows to be impacted by changes
ImplementationQuestion	<ul style="list-style-type: none"> ● question for magnitude of change ● relative or absolute change ● defines domain of possible inputs by user ● defined for a specific solution ● same question can be used by multiple solution parts
PossibleImplementation-Area	<ol style="list-style-type: none"> a. question, where a solution can be implemented b. defines boundaries where users can draw in their implementation area c. defined for a specific solution d. same area can be used by multiple solution parts

In the workshop mode, the defined solutions can be picked and applied to the strategy of the user. The database tables in the backend, that store the input are mapped in the "user inputs" box in the UML-diagram in Figure 11 and described in Table 3.

Table 3: Description of classes holding inputs of workshop participants

CLASS	DESCRIPTION
Strategy	<ul style="list-style-type: none"> ● auto-created for each user who has access to the case study ● key flow specific ● at the moment restricted by the logic to one per user and key flow

	<ul style="list-style-type: none"> ● can contain multiple implementations
SolutionInStrategy	<ul style="list-style-type: none"> ● single implementation of a solution by the user
ImplementationQuantity	<ul style="list-style-type: none"> ● “answer” to ImplementationQuestion ● concrete values for magnitude of direct changes made to flows
ImplementationArea	<ul style="list-style-type: none"> ● “answer” to PossibleImplementationArea ● geometry to filter actors whose incoming resp. outgoing flows will be changed, depending on definition of FlowReference

Data Access

The resources which are stored in the backend are accessible via an API following the specifications of RESTful APIs. Every type of resource is represented by an own route that is accessible via HTTP methods. The available methods and their corresponding actions are displayed in Table 4.

Table 4: HTTP methods, the actions performed on request and the permissions required to execute the corresponding action

HTTP method	action performed	required permission
GET	get resource	view
POST	create resource	add
PUT	update resource (create if not exists)	change (add)
PATCH	partially update resource	change
DELETE	delete resource	delete

Unlike REST standards, the implemented API is not perfectly stateless. The user session stored in the backend is used to determine the permissions to access resources. The access to specific case studies is set in the profile a user has (see [Data Structure](#)). Furthermore, the actions a user can perform with different types of resources are regulated in detail via user and group permissions.

The actions to view, add, change or delete a resource are prohibited by default and have to be permitted individually. An exception is the administrator, who has access to all resources with all actions by default. Next to the permissions for access to resources there are two special permissions on a case study in the login-section. These permissions determine, if a user or group is permitted to enter the setup mode respectively has access to the data entry section.

Regardless of the permissions, most of the resources are protected against cascaded deletions. E.g. the “Activity” model as shown in the code sample in Figure 6 is a protected

resource that can only be deleted in the API, if there are no flows or other resources left, that point to the Activity. The protection does not apply to the database tables themselves and can be circumvented by deleting the entries there directly.

The permissions can be defined for individual users or for a group of users. It is recommended to create groups with special permissions and assign the users to these groups. It could become very time consuming and confusing to set permissions to every single user. The groups defined in the REPAiR-project and their rights assigned are described in Section 4.5.

An example for a workshop group is shown in Figure 12: Set up page of the permissions of the group “WorkshopParticipant” in the Django administration site. The groups used in the REPAiR project can be found as a JSON-dump at `.\repair\fixtures\repair_groups.json` with the relations to permissions as natural keys (in plain text).

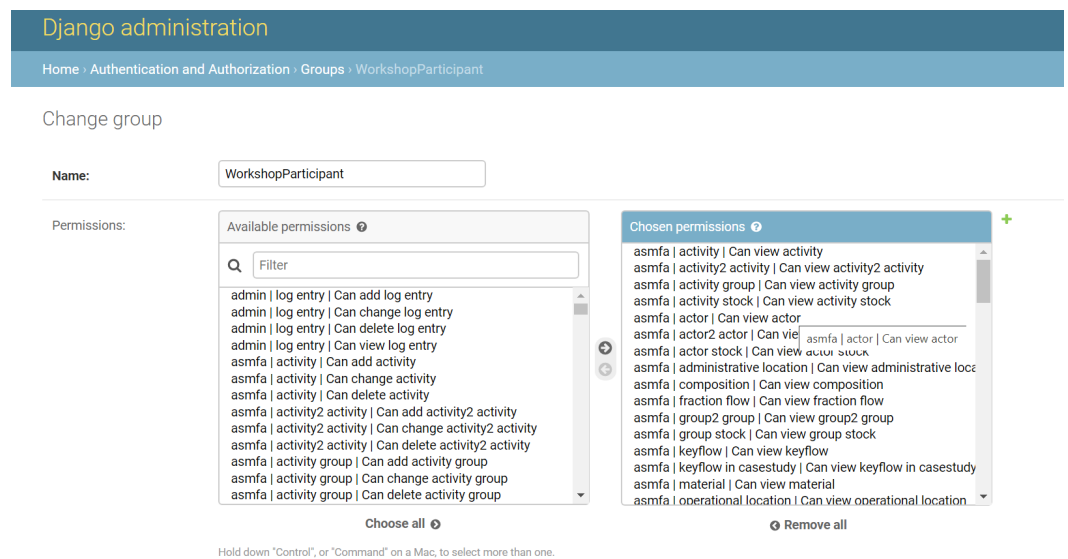


Figure 12: Set up page of the permissions of the group “WorkshopParticipant” in the Django administration site

The routes to the resources are defined in `.\repair\rest_urls.py`. The entry point to the API is mapped to `<domain>/api`. It can be navigated through HTML views if called in a browser.

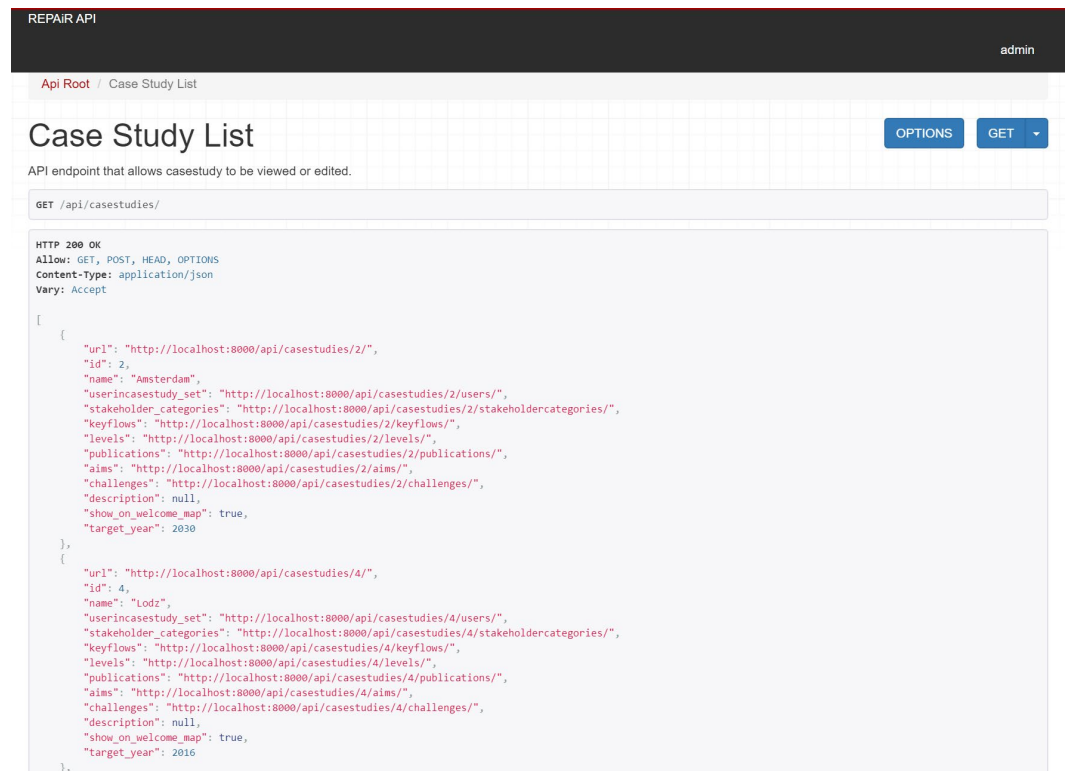


Figure 13: Screenshot of the HTML list-view on available casestudies
<https://gdse.h2020repair.bk.tudelft.nl/api/casestudies/>

The List-View shows a list of all available resources. For example `<domain>/api/casestudies/` shows all case studies and their attributes including the id (see figure 13). The details to the case study with id 1 can be requested via the route `<domain>/api/casestudies/1`. The routes represent the hierarchy and relations of the underlying models. Deeper routes always have a relation to its preceding subroutes. For example, activities available in case study 1 can be accessed via `<domain>/api/casestudies/1/activities`.

Which case study an activity belongs to, is defined by the key flow of its parent activity group as defined in the Activity serializer (see code sample in Figure 7:).

A detailed documentation of all routes and available methods can be obtained under `<domain>/api/docs`.

The screenshot displays the REPAiR API Documentation interface. On the left is a dark sidebar with a menu listing various API categories: areasofprotection, casestudies, impactcategories, keyflows, materials, processes, products, publications, reasons, sustainabilities, targetspecialreference, targetvalues, and wastes. The main content area is titled 'REPAiR API Documentation' and 'areasofprotection'. It lists four endpoints: 'list' (GET), 'create' (POST), 'read' (GET), and 'update' (PUT). Each endpoint includes a brief description, a table of path parameters, and a code block for interacting with the API using the 'coreapi' command-line client. For example, the 'list' endpoint has a description 'Check if user is permitted for list view.' and a table with no parameters. The 'create' endpoint has a description 'Check if user is permitted to create this object.' and a table with two parameters: 'name' (required) and 'sustainability_field' (required). The 'read' endpoint has a description 'Check if user is permitted for detail view.' and a table with one parameter: 'id' (required). The 'update' endpoint has a description 'Check if user is permitted to update the object.' and a table with no parameters. Each endpoint also has an 'Interact' button and a code block showing the command to use the 'coreapi' client.

Figure 14: Screenshot of the REPAiR API Documentation <https://gdse.h2020repair.bk.tudelft.nl/api/docs/>

All information about the request and the expected response is encoded in the URL-route and in the query parameters. Usually a GET request is sent to the URL to view resources with filter instructions inside the query parameters. Due to technical limitations, URLs might get too long especially when filtering spatially. Therefore, some backend views like actors and locations are configured to accept query parameters in the request body as a substitute for the URL query parameters. To indicate a view request with query parameters inside the body, a POST request has to be sent to the resource route alongside the URL parameter “GET=True”.

To avoid the time-consuming individual upload of resources of the same type in large amounts, the possibility of bulk uploading data was implemented (REPAiR, 2020).

Therefore some list-views of the REST API provide the possibility to upload a file containing multiple entries including the values of the fields to be set (e.g. `/api/casestudies/<id>/keyflows/<id>/activitygroups/`). A template file with the required structure to fill and upload can be requested by sending a get request to the list-view of resources supporting bulk uploads with the query parameter “request” set to “template” (e.g. `<domain>/api/casestudies/<id>/keyflows/<id>/activitygroups/?request=template`)

Flow Filters

As mentioned in the chapter [Data Structure](#), the data of flows is stored in the database on actor level only. To view flows on activity or activity group level the data has to be aggregated. The aggregation is done server-side. Furthermore, the flows can be filtered by rather complex, chained filters. That includes filters for the origin or destination of flows and the materials (see table 5). The filter functions can be found in the “asmfa” app under `.\repair\apps\asmfa\views\flowfilter.py`.

Table 5 parameters for filtering and aggregation flows

KEYWORD	DESCRIPTION	ATTRIBUTES	ATTRIBUTE DESCRIPTION
filters	list of logically linked sub-terms (AND linked)	link	logical link between django filter functions in sub-term value: and/or
		<django filter function>	django field lookups (see https://docs.djangoproject.com/en/3.1/topics/db/queries/)
		<django filter function>	
		[...]	
	e.g. filters: [{ link: or, origin_id_in: [1,5], destination_id_in: [1,5] }, { link: and, amount_gt: 10, amount_lt: 100 }]		Filter all flows, where the origin-id is 1 or 5 or where the origin-id is 1 or 5 and among the remaining flows, filter the flows with an amount between 10 and 100 (greater than 10 AND less than 100).
materials	filter and aggregate by materials	ids	list of material ids the flows should contain, this includes flows with descendant materials of given ancestor materials
		aggregate	aggregates descendant materials to given ancestor materials (attribute “ids”), aggregates to top level materials if attribute “ids” is empty value: true/false
		unaltered	list of material ids that are excluded from the aggregation
		e.g. materials: { ids: [1], unaltered: [10, 15], aggregate: true }	
aggregation_level	aggregate flows on origin and destination side from actor level to given level	origin	aggregate origins of flows value: activity/activitygroup, stays on actor level if left empty
		destination	aggregate destinations of flows value: activity/activitygroup, stays on actor level if left empty
		e.g. aggregation level: { origin: activity, destination: activitygroup }	

The route to the flow filters is `<domain>/api/casestudies/<id>/keyflows/<id>/flows`. The filter parameters shown in Table 5 can either be put in the query string as query parameters or in the body with a POST request and the query parameter “GET” set to true.

In addition, the query parameter “strategy” can be added. The strategy-parameter defines the id of a *user strategy* to show the flow data resulting from the impact of this strategy, which has to be calculated by the [Graph Walker](#) algorithm beforehand. If the “strategy” parameter is not set, the status quo data is shown.

Indicators

Indicators are used to “quantitatively assess the key waste flows [...] in relation to their geographical context” (REPAiR, 2020). They sum up flow amounts spatially after filtering the flows. Unlike the [Flow Filters](#), the filter parameters are not solely passed by query parameters. Instead, an indicator with filter settings has to be defined beforehand by posting the definition to the route

`<domain>/api/casestudies/<id>/keyflows/<id>/flowindicators/`.

The indicator definition is stored in the database. The corresponding model “FlowIndicator” can be found in the file `.\repair\apps\status_quo\models\indicators.py`. Next to general meta information the FlowIndicator holds information about its type.

There are four different types of indicators at the moment:

- “IndicatorA” - the amount of a single flow filter
- “IndicatorAB” - the ratio between two separate flow filters
- “IndicatorInhabitants” - the amount per inhabitant
- “IndicatorArea” - the amount per area (ha)

The filter settings are held within an “IndicatorFlow”. Up to two IndicatorFlows can be set to a single FlowIndicator, depending on the indicator type. Defining two IndicatorFlows is only reasonable for the type “IndicatorAB”.

The name of the indicator type that is set to the FlowIndicator has to match the names of the computation classes that are located in

`.\repair\apps\status_quo\models\computation.py`.

Having a defined indicator, the computation can be triggered by sending a request to the route `<domain>/api/casestudies/<id>/keyflows/<id>/flowindicators/<id>/compute` with third id being the id of the created indicator. All computations filter the flows first and aggregate them based on the spatial settings. The indicator type-specific calculations are done last.

By default, the indicator computation is done for a single area, either the focus area or the case study, whichever spatial reference is set to the FlowIndicator. To calculate the indicator for specific areas, the IDs of the areas can be passed with the query parameter “areas” to the computation route. To get a single value, the query parameter “aggregate” has to be set to true. If the parameter “aggregate” is set to false, the response contains for each area a separated amount. Similar to the [Flow Filters](#), indicators can be calculated for a user-defined strategy by setting the “strategy” query parameter to the id of the strategy. Otherwise the data of the status quo is used for the calculation.

Graph Walker

The Graph Walker calculates the impact of strategies on the status quo flows.

As the name suggests, the Graph Walker works on a graph representation of the flow data. The actors are represented by vertices connected by edges representing the flows. Therefore, the status quo data has to be translated into the so-called base graph. Every flow of material between actors becomes a separate edge with the information about its direction, material and the amount of material. The resulting graph is directed with parallel edges (see figure 15).

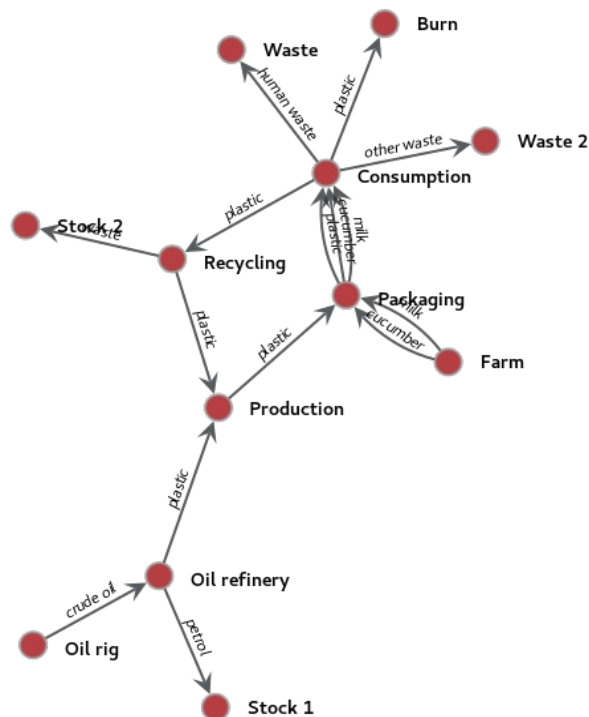


Figure 15: Example graph for material flows between actors

There are no automatic triggers to build the base graph on change of data due to performance reasons. The translation has to be triggered key flow-wise manually. This should be done per key flow in preparation of the workshops when the status quo flow data is not supposed to change anymore.

To trigger the build, a GET request has to be sent to `<domain>/api/casestudies/<id>/keyflows/<id>/build_graph/`. After the graph has been built, it is stored in a local graph file on the server named after the key flow and case study. The previous base graph will be overwritten.

Ideally, the graph is balanced. This means, that the sum the amounts of all incoming flows at each node matches the sum of the amounts of the outgoing flows. If the graph is balanced, the ratio between the input and output (called “balance factor”) is exactly 1.0 for each node, except for sources, which have no in-flows, and for sinks, which have no out-flows.

In reality this is unlikely to achieve. The balance factor for each node is stored in the base graph as it is needed later during the calculation of the strategy impacts.

With an already calculated base graph in place, the calculation of the impacts of a strategy with a *strategy_id* can be triggered by sending a GET request to `<domain>/api/casestudies/<id>/keyflows/<id>/strategies/<strategy_id>/build_graph/`.

To simplify the algorithm, the calculations of the solutions in the strategy are not done in parallel at the same time but for one solution after the other. The order is determined by the attribute *priority* of the *SolutionInStrategy* objects. The order can be set by the workshop users by ordering the solutions with drag and drop. The same applies to the solution parts within a solution. They are calculated in the order set by their *priority* attribute. The priority is defined frontend-side by ordering the parts in the setup mode. The order of the calculations has an impact on the results and should be considered when defining the solution parts.

Before the actual calculation of a solution part begins, the data is prepared for the calculation. The flows that will be changed directly, named *implementation flows*, are filtered according to the settings of the *FlowReference* object behind the *flow_reference* attribute of the solution part (see Table 5) and the *implementation areas* drawn by the users.

In the next step the graph walker algorithm figures out, how the implementation flows are changed by the solution part.

There are six schemes available for the solution part

- Modify existing flow
- Shift origin of a flow
- Shift destination of a flow
- Create new flow
- Prepend flow
- Append flow.

The information is encoded in the scheme attached to the solution part. A detailed explanation of the definitions of the schemes and their effects is available in the Deliverable 2.5 (p. 45).

The change to the amounts of the implementation flows are specified by the value of the `ImplementationQuantity` defined by the workshop user. The `ImplementationQuestion` defines, if the changes are absolute or relative to the status quo amount of each implementation flow.

Information about other changes of attributes (e.g. the material) of the implementation flows is held by the `FlowReference` object related to the *flow_changes* attribute of the solution part (see Table 5). The determined changes are applied to the edges which represent the implementation flows in the base graph. New edges are added to the graph in case when new flows were created.

Next, the changes need to be propagated through the graph. This is where the actual Graph Walker algorithm comes into play. The Graph Walker algorithm only takes into account the *implementation flows* and the *affected flows*. All other flows in the base graph are ignored.

The definitions of *affected flows* make the following traversal through the graph more performant, because the number of nodes and edges are reduced to the relevant ones. In addition, the definition of *affected flows* allows to specify, which flows and materials are affected by a solution. For example, if in the graph in Figure 15: Example graph for material flows between actors' the plastic packaging of cucumbers is reduced, then the amount of recycled and burned plastic will be reduced, but not the other waste fractions. Therefore, the recycled and burned plastic can be marked as *affected flows*.

In a first run the Graph Walker traverses the graph in a breadth-first-search manner. It starts from each implementation flow, starting at the target node of the implementation flow (the *implementation edge*). When for example the amount of the implementation flow is increased by 1.000 tons, the graph walker algorithm propagates this change of 1000 tons in the graph.

When visiting a node, the targeted change of the implementation edge is distributed to the outgoing flows. The 1000 tons are distributed to all out-flows of this nodes, that are *affected flows*. The Graph Walker algorithm takes the balancing-factor (described above) into account. This is continued until all nodes are visited.

Then, the algorithm goes "upstream" from the implementation edges, until all nodes are visited in this direction.

By default, the Graph Walker actually goes upstream first and downstream second, because “demand dictates supply”. This is technically implemented in the Graph Walker algorithm by temporarily changing the direction of the graph representation while propagating changes upstream.

The algorithm attempts to keep the balance factors at all vertices intact while distributing the changes upstream and downstream in a graph with circular loops. This might have the effect, that the actual value at the implementation edge might mismatch the targeted value after the first iteration of the algorithm. Therefore, in the next iteration the algorithm tries to reduce the mismatch between the targeted change at the implementation edge and the calculated change.

In each iteration, the graph will be traversed upstream and downstream again and the delta to the targeted value is distributed in a similar way to the first iteration. This is done until the mismatch is within a defined tolerance, or the maximum number of iterations is exceeded.

After all implementation flows of all parts of the solutions are processed, the changes to the base graph produced by the strategy are saved in the database. New flows are stored in the FractionFlow table, with a relation to the strategy they are created by. Status quo flows have no relation to any strategy. Changes to flows are stored in the StrategyFractionFlow table. The StrategyFractionFlow holds all changes to attributes like the materials, processes and the new amounts calculated for a strategy. To indicate, that an attribute is not changed by the strategy, its value is set to NULL.

When the strategy is calculated, its results can be requested via the flow filter route ([Flow Filters](#)) as well as via the indicator route ([Indicators](#)) of the API by passing the query parameter “strategy” with the ID of the strategy.

3.2 Web frontend

The frontend of the GDSE serves as a view on the data in the server backend and enables the interaction of the user with the data. It is implemented as a web site and can be rendered in any modern browser that supports HTML5 and JavaScript ECMAScript 5. The communication with the backend is performed with HTTP (see [Data Access](#)).

This chapter focuses on the technical details of the implementation of the frontend. How to use the web frontend of the GDSE is described in detail in the Deliverable 2.5 (REPAiR, 2020) and here: [Using the GDSE in Online-Workshops'](#)

Architecture of the web frontend

As mentioned in chapter [Basic architecture](#) the HTML templates for the web pages are filled and served by the Django backend. During the project, it became clear this approach is not dynamic enough on the client side. Big parts of the web pages need to be replaced dynamically, based on the user input. To achieve that with client-side views, the frontend received its own independent MVC (Model-View-Controller) implementation, Backbone.js. This results in some unusual compromises like the mix of different template languages in the same template. The router function remains solely in the backend.

Backbone.js is a JavaScript library with support of RESTful interfaces and suitable for the implementation of single-page applications. In our case, every GDSE step (Study Area, Status Quo, Targets, Strategy, Conclusion) is designed as a single-page.

The communication and the serialization/deserialization are realized with Backbone models and collections. A model represents a single resource. Collections hold a number of models of the same type, representing a list view of resources in the Rest-API. The basic models and collections are adapted to the needs of the django backend (support for file uploads, uploads of resources as forms).

Models and collections have to be allocated via the *new*-Operator and then changed or deleted and synchronized with the Rest-API (see code sample in Figure 16). To simplify the use, a tag and the IDs of the preceding resources can be passed instead of the complete URL. The tags and corresponding URLs can be found in the `.\repair\js\app-config.js` (see Figure 17). The URLs in the `app-config.js` have to match the URLs of the Django router (see [Data Access](#)).

```
this.stakeholderCategories = new GDSECollection([], {
  apiTag: 'stakeholderCategories',
  apiIds: [ _this.caseStudyId ]
});
[...]
```

```
_this.stakeholderCategories.create({name: name}, {
  success: _this.initStakeholders,
  error: _this.onError,
  wait: true});
```

Figure 16: Code sample - use of collections - creation of a stakeholder category `.\repair\js\views\study-area\stakeholders.js`

```
/** urls to resources in api
 * @name api
 * @memberof module:config
 */
config.api = {
  base:                '/api', // base Rest-API URL
  casestudies:         '/api/casestudies/',
  [...]
  stakeholderCategories: '/api/casestudies/{0}/stakeholdercategories/',
  stakeholders:
    '/api/casestudies/{0}/stakeholdercategories/{1}/stakeholders/',
  [...]
};
```

Figure 17: Code sample - excerpt of routing config in `.\repair\js\app-config.js`

The communication with the backend is asynchronous to avoid blocking calls and keep the web site responsive. For this purpose, success and error functions have to be defined and passed to the calling function. They are called, when the asynchronous call is completed, and the response arrives at the client again.

The Backbone views serve mostly as the Control of the MVC pattern. It listens to the user input, interprets them and calls the model classes, other views and renders templates if necessary. A basic view to derive custom views from is provided at `.\repair\js\views\common\baseview.js`. It holds basic render functions and general auxiliary functions.

The templates take the role of the View within the MVC. The template engine uses client-side `js-underscore`. The *underscore templates* are wrapped in the HTML provided by *Django*, and tagged as scripts (see code sample in Figure 18). To load a template into a view and to render it, access the template via its id and get the inner html in order to pass it to the template engine (see code sample in Figure 19). The resulting HTML can be put into existing containers.

```

<script type="text/template" id="conclusion-item-template">
<div class="row" style="height: 100%;">
  <div class="col-md-7 bordered" style="height: 100%;">
    <%= conclusion.get('text') %>
  </div>
  <div class="col-md-1 bordered" style="height: 100%; overflow: hidden;">
    
  </div>
  <div class="col-md-2 bordered" style="height: 100%; overflow: hidden;">
    <label><%= section %></label>
  </div>
  <div class="col-md-2 bordered" style="height: 100%; overflow: hidden;">
    <label><%= conclusion.get('step') %></label>
  </div>
  <button name="remove" class="btn btn-warning square" title="remove
conclusion" style="position: absolute; right: 10px; top: 10px;">
    <span class="glyphicon glyphicon-minus"></span>
  </button>
</div>
</script>

```

Figure 18: Code sample - embedded script of an underscore template inside the django template
 .\repair\templates\conclusions\workshop.html

```

addConclusionItem: function(grid, conclusion){
  var _this = this,
  item = document.createElement('div'),
  html = document.getElementById('conclusion-item-template').innerHTML,
  template = _.template(html);
  item.innerHTML = template({
    conclusion: conclusion,
    section: this.sections.get(conclusion.get('section')).get('name')
  });
  [...]
  grid.add(item);
  [...]
}

```

Figure 19: Code sample - add a row showing a conclusion by rendering the template from code sample in
 Figure 18 \repair\js\views\conclusions\conclusions.js

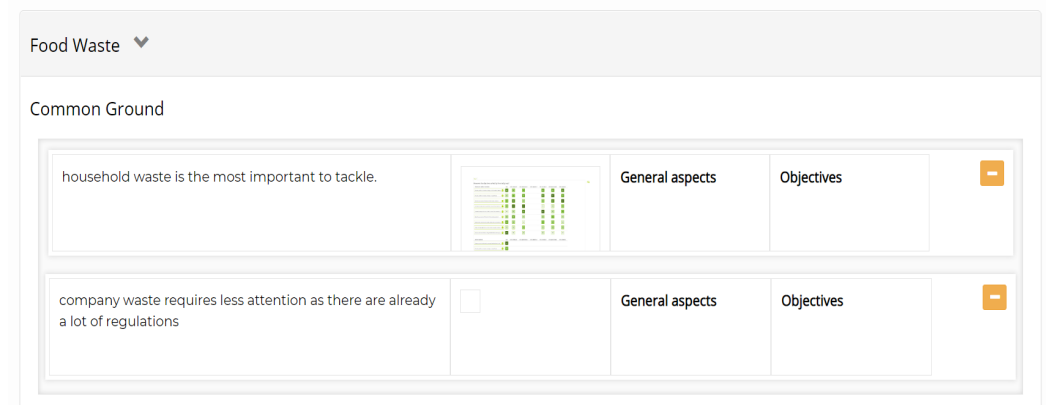


Figure 20: Two rendered conclusion rows in the GDSE as a result of code sample in Figure 19

Design

The design of the website is primarily defined by the HTML. Some of the styles of the HTML documents are described inline but most general styles are outsourced to CSS-files located in `.\repair\static\css`. The external stylesheets can either be imported in the header of the Backbone views or inside the HTML files.

The most important stylesheet file is `.\repair\static\css\base.less`. It defines the basic appearance of the web site. To be able to define reappearing attributes like the main colours in variables, the basic styles are written in LESS, a dynamic pre-processor style sheet language. Webpack is configured to compile the LESS-files to CSS with a less-loader to be interpretable by the browser.

The GDSE website is designed to be responsive. Even though the GDSE is mainly used on big touch screens with similar proportions during the workshops, its responsive web design aims at supporting a variety of resolutions and devices such as desktop PCs and tablets (see Figure 21).

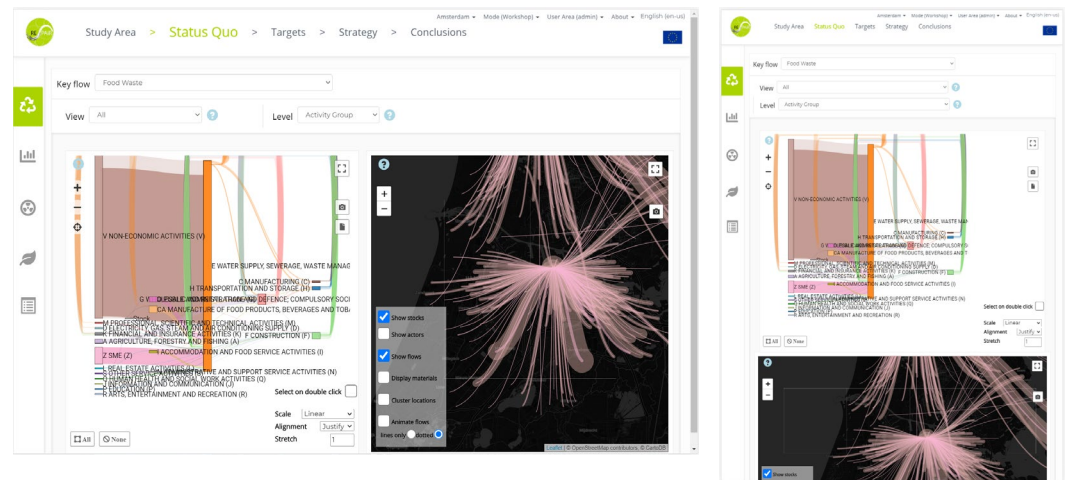


Figure 21: Layout of same page at different resolutions. left: laptop with HiDPI screen (1440x900), right: iPad (768x1024)

The responsive design is realized with the JS/CSS/HTML library [Bootstrap](#). Bootstrap ships with its own collection of stylesheets and matching scripts. Bootstrap's HTML layout is grid-based and organized in rows and columns. Most of the elements of the GDSE pages follow this grid pattern with breakpoints for medium resolutions such as in tablets. Below those resolutions ($\leq 768\text{px}$) the layout breaks and columns are displayed vertically (see Figure 21).

The assignment of containers to the grid is done via class names. Same applies to the other utilized Bootstrap elements like the navigation bars and buttons. The various GDSE dialogs for user inputs and displaying messages are also built with Bootstrap and controlled by the views. To avoid a generic look and to achieve a more unique appearance, the styles of the Bootstrap containers are customized for the GDSE by overriding respectively extending them with CSS files (e.g. `.\repair\static\css\main-navbar.less`)

The layout of the GDSE site follows the logic of the decision process with its five steps. The basic page layout including the main menu is set by the basic template `.\repair\templates\base.html`. Each step is implemented as a single page and can be accessed via the corresponding item of the main menu. Every step is split into sub-steps, and are also referred to as screens. The sub-steps are navigable via the side menu. The `base.html` only provides the container for the side-menu and imports. Its items have to be filled in the templates of the steps which extend the `base.html`. The icons used in the sidebar and in the buttons are taken from the Bootstrap glyphicon collection and from the free version of Font Awesome (<https://fontawesome.com/>).

Switching between sub-steps is implemented as tabbed interfaces with Bootstrap. The contents of the sub-steps are put in by the views into separate tab-containers on the same page. Only the tab of the active sub-step is visible. Switching a sub-step in the side-menu activates another tab and makes it visible while hiding the previously active tab. This way switching tabs is very fast as no content has to be loaded on change. The downside is that the whole page has to be rendered with all sub-steps and their data simultaneously even if some of the sub-steps will not be accessed.

JS Entry Points

The main scripts for each page are located at `.\repair\js`. The templates of the pages import their associated script which are named similarly e.g. the page `<domain>/study-area` imports `repair/js/study-area.js`.

These scripts serve as entry points for all subsequent scripts. The main script loads data, libraries and stylesheets which are shared by views and finally calls the views, usually the views that control a single sub-step. The views themselves load further data and libraries and call further views on demand like visualizations. The imports within the scripts are realized with `require.js` (<https://requirejs.org/>).

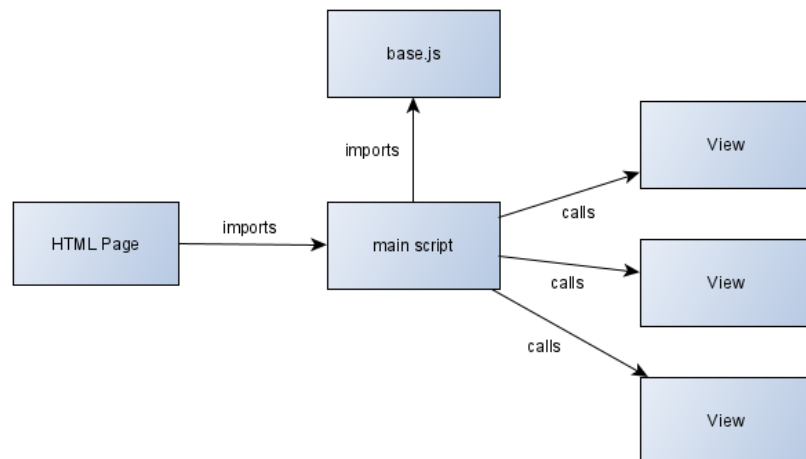


Figure 22: Entry point to the JS scripts on a page

In addition, the file `.\repair\js\base.js` has to be imported by either the step-templates or by the entry points. The script `base.js` is loading all basic requirements and stylesheets and is overriding functions as needed by other entry points.

Sessions

The Django session usually only contains the session ID, the CSRF token and the language code. The user is identified by the session id after logging in. The session is stored in cookies.

Django is configured to require a token for POST requests to protect the web site against CSRF attacks where an attacker rides the session of the logged-in user. Therefore, the token has to be put into all POST forms within the Django templates with the tag `{% csrf_token %}`. Otherwise the POST requests would be rejected. Backbone doesn't support CSRF by default. The underlying AJAX has to be set up to send the token inside the header. This is implemented in the *base.js* script (see [JS Entry Points](#)). It has to be imported to get the persistence methods of the Backbone models to work with the backend.

The more complex inputs of users like strategies and solutions have corresponding models on the backend side to store them in the database. To make it more flexible, additional session information can be added in the JS scripts dynamically as attributes of the session object without the need of making changes to the backend. The session can be accessed via the config object (*.\repair\js\app-config.js*). Amongst other things this is used to store the selected case study, the current mode (setup or workshop mode) and the layers the user selected on the study area map.

The session can be persisted by posting the attributes as a JSON to the route `<domain>/session`. They are stored serialized as a JSON string in the Profile of the user. The same route is used to retrieve the stored session attributes. The requests can be called with the functions `fetch` and `save` of the session object. The persistence of the session was implemented to allow cross-platform sessions. This way the user can keep his settings even when switching between different devices.

Visualizations

While the data is stored in and calculated, filtered and aggregated on demand by the backend, the frontend is responsible for the visualization of the data. This includes displaying maps and layers, generating bar charts for the indicators and Sankey diagrams for the flow data.

Most of the maps in the project, such as the study area map and the indicator map are realized with the JavaScript library [OpenLayers](#) that uses an HTML5 canvas for drawing the data. It was extended to meet the requirements of the GDSE with simplified feature-, layer- and geometry-management (`.\repair\js\visualizations\map.js`).

An exception is the map used to visualize the material flows spatially. It is based on the JavaScript library [Leaflet](#). Instead of utilizing a canvas, it draws the background tiles as images and uses SVG-containers as overlays. The basics of the visualization of the flows on a map were developed for the GDSE as part of the master thesis on the development of a visualization concept for the representation of geo-referenced flows by Jochim (2018).

To integrate the code implemented as part of the master thesis into the GDSE, some adaptations were made (`.\repair\js\visualizations\flowmap.js`). The material flows between the same origin and destination locations were originally displayed as parallel flows with common arrowheads indicating the direction of the flow (see figure 23). While applying this to sample data the arrowhead proved to be hardly visible in case of flows with relatively small amounts. The bundled coloured materials were also hardly distinguishable and difficult to hit with the mouse cursor to show the tooltip. This was changed into bundling the materials by default and listing them in detail in the tooltip when hovering the bundled flow. On demand, the materials can be visualized as easier distinguishable separate but not parallel lines with a bigger bounding box (see figure 24). The directions of the flows are visualized as animations moving from origin to destination. Furthermore, the flow map was extended to display stocks as pie charts on the map and to cluster actors based on the zoom level.

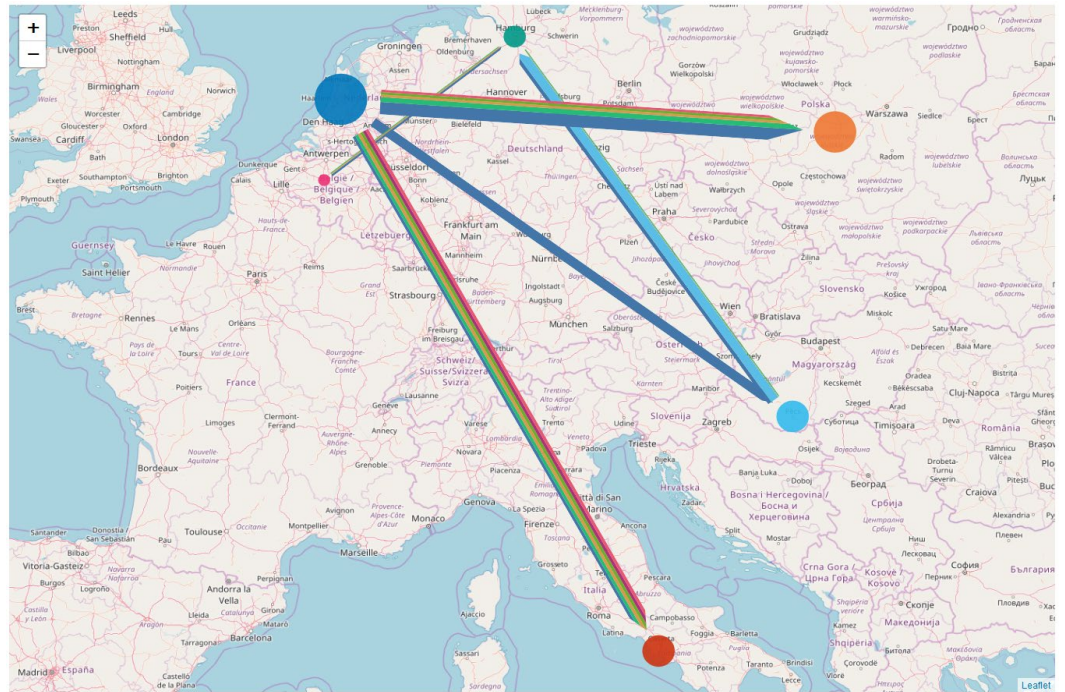


Figure 23: Visualization of flows with OSM background map (Jochim 2018)

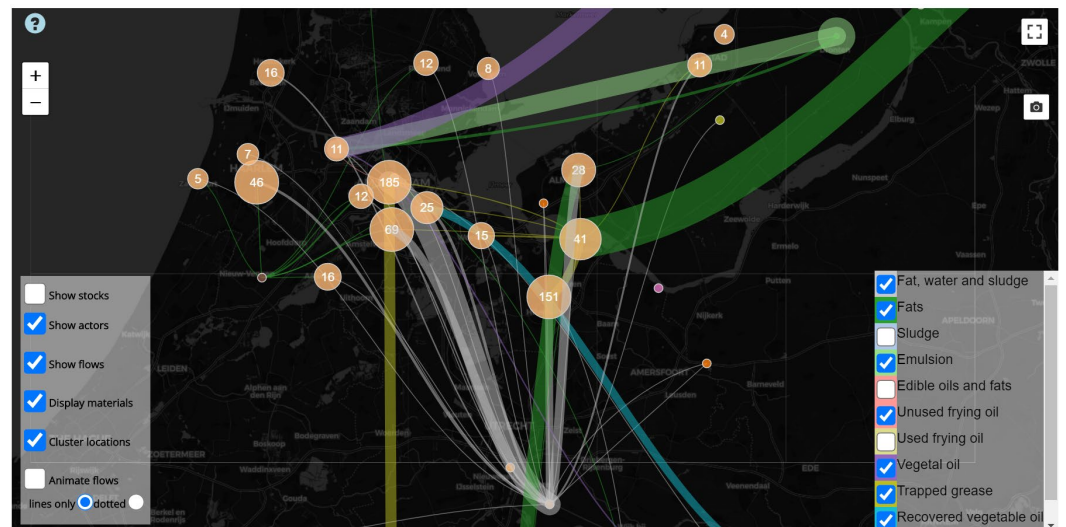


Figure 24: Visualization of flows in the GDSE with overlay controls

The flow map is combined with a Sankey diagram in a reusable Backbone View to be found at `./repair\js\views\common\flows.js`. It registers event listeners to the diagram, requests specific filtered flows from the backend on click and displays them in the flow map.

Like most of the diagrams in the GDSE, the Sankey diagram is realized with the library [D3.js](https://d3js.org/) (<https://d3js.org/>). To be able to display circular flows, it is based on the D3 circular Sankey

extension taken from <https://gist.github.com/cfergus/3956043>. The further extended implementation for the GDSE (.js\visualizations\Sankey.js) features a zoomable and movable container, selectable flows, customized tooltips and customizable layout. A screenshot of flows rendered with the GDSE Sankey implementation and an overlay to control the layout can be seen in Figure 25.

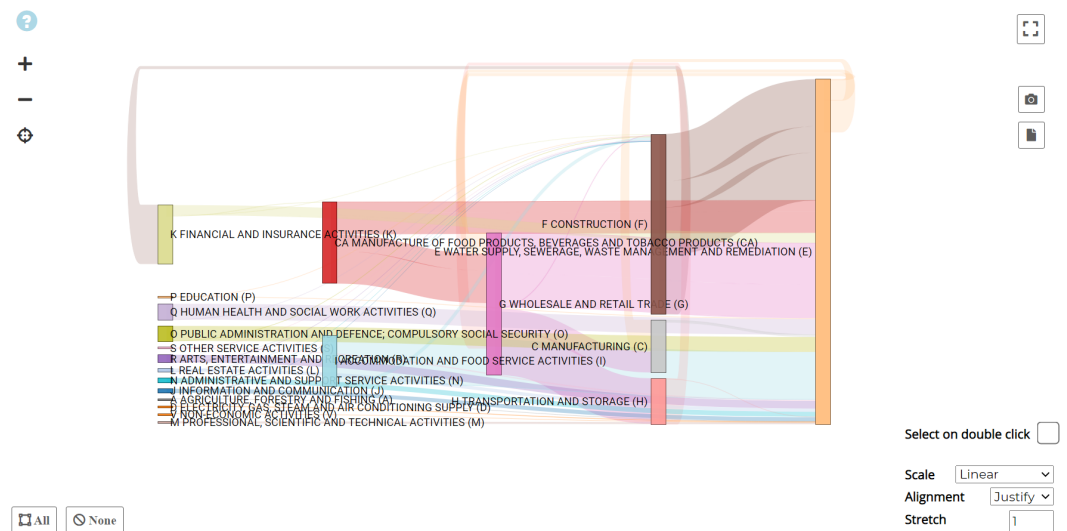


Figure 25: Sankey diagram with overlay controls

All visualizations support touch control. Like in the rest of the website most elements are touch-enabled by default by the browser translating the touch gestures like dragging and tapping to mouse inputs. Some of the default interactive elements like buttons and checkboxes had to be upscaled via CSS to be easily pressable. OpenLayers and Leaflet have touch support for more elaborate touch gestures already built in. Only a few adaptations had to be made.

To make HTML containers draggable by touch or mouse inside a grid layout, the JavaScript library [muuri](https://muuri.dev/) (<https://muuri.dev/>) was used. An example is shown in Figure 26.

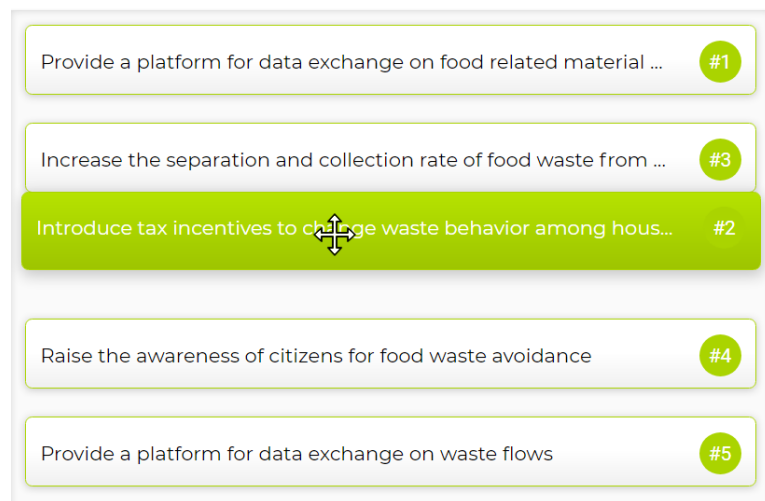


Figure 26: Muuri container, draggable and movable by touch in the GDSE “Ranking Objectives”

3.3 Internationalization

REPAiR is an international project. The workshops take place in different countries across Europe. So, the content of the GDSE has to be delivered to a potentially non-English speaking audience.

The language can be changed client-side in the browser and is stored in the session object with a language code like “nl”. The currently available languages correspond to the countries, the workshops take place in: Dutch (“nl”), Polish (“pl”), Hungarian (“hu”), Italian (“it”), German (“de”) and English (“en-us”) as the default language. To support the internationalization, preparations have to be made while implementing the backend and frontend modules.

All texts in the code should be written in English first because the default language is set to English. To mark the strings that are potentially visible to the user for translation, they have to be wrapped in special functions. Those functions or annotations depend on the programming language as shown in Table 6.

Table 6: Annotations to mark strings to be translated and the imports required to use the annotations in the different programming/markup languages used in the project

LANGUAGE	REQUIRED IMPORT	ANNOTATION
HTML	{% load i18n %}	{% trans “MyString” %}

Python	from django.utils.translation import ugettext as _	_("MyString")
JavaScript	-	gettext("MyString")

If the template engine or the interpreter comes along an annotated string at runtime, it looks the string up in a file containing all translations corresponding to the current language code of the session. If a not-empty entry is found, the string will be replaced by it.

To create a language file, you have to execute:

```
python manage.py makemessages -d djangojs -l <code> -e html,js,py
```

where `<code>` has to be replaced by an actual language code like "hu" (without brackets). The command will parse all annotated strings and collect them in the file *djangojs.po* in the directory `./repair/locale/<code>/LC_MESSAGES`.

The file is human readable, so the input of the translations can be done with a text editor. There is also specialized software like [Poedit](https://poedit.net/) (<https://poedit.net/>) to do this. The translated file has to remain in the exact same spot where it was created to be found by the running system. The translation should be done directly in this file, or - if the translation is done externally - simply overwrite the *djangojs.po* file afterwards. Then the changes need to be pushed and merged into the productive branch.

Not only texts but numbers should be localized, because on German you write 1.234,56 €, while in English, you would write 1,234.56 €. The backend provides numbers in English number format. To localize the numbers in the frontend views to the language settings, the "format" function of the base view (`./repaird/js/views/common/baseview.js`) can be used. Alternatively, the numbers can be localized by calling the `toLocaleString` function with the language code as an argument e.g. `value.toLocaleString(session.language)`.

4 Data Management

4.1 Overview of the required data and user input

There are three different kind of data in the GDSE to consider:

- Data that is generated outside of the GDSE and uploaded on the GDSE for the purpose of its exploration and visualization;

- Data that is generated outside of the GDSE but gets analysed using the GDSE this way providing new insights and producing new datasets;
- Data that is collected using the GDSE by allowing user input.

Therefore, the data can be entered into the GDSE by:

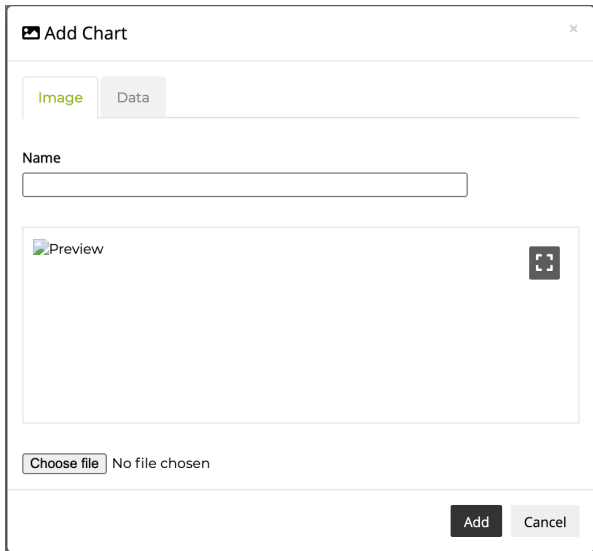
- a *data captain* who prepares data for the analysis in the GDSE;
- a specialized *researcher*, e.g. LCA expert who collects and analyses data outside the GDSE;
- a *PULL leader* in preparation of the workshop;
- a *workshop participant* who enters the data during the workshop as asked by the workshop moderator.

Types of data that are required to be uploaded into the GDSE are described here below following the five steps of the GDSE.

Table 7: Overview of the required data and user input

GDSE Step & Substep	Study Area: Maps
Data purpose	Maps are meant for collecting and visualizing static data about the study area which means that the data portrayed here does not represent processes, scenarios or relationships but a fixed state of elements related to the social, environmental and infrastructural context of the study area. During the workshop participants are allowed to explore different map layers in various combinations, zoom in and out to the chosen scale.
Data type	Geospatial data
Data formats	WMS Service
Data upload procedure	Data is uploaded directly on GeoServer as explained in the section 4.4. <i>Integrating Geodata and Maps via WMS/WFS</i> Maps can be grouped in categories and interactively rearranged in order by the interface users. The system also allows adding publicly available WMS/WFS layers that are useful as e.g. background maps.

Special requirements	Maps need to be provided along with the layer style and a relevant legend.
Updating data	Maps can be updated by reuploading them on the GeoServer and again loading into the GDSE

GDSE Step & Substep	Study Area: Charts
Data purpose	Charts are meant to display visual information about the study area that is not available in the form of maps
Data type	png, gif, jpeg
Data upload procedure	<p>Images are uploaded directly through the user interface by selecting +Chart button and giving a relevant name to the image. Images can be grouped in Categories.</p> 
Special requirements	None
Updating data	Charts can be updated by removing the image and uploading a new one

GDSE Step & Substep	Study Area: Stakeholders
Data purpose	A list of stakeholders organized into categories and their descriptions. The list is later used to select stakeholders that can be involved in the implementation of

	specific eco-innovative solutions.
Data type	Text
Data formats	-
Data upload procedure	Typed using user interface, cannot be uploaded in bulk
Special requirements	None
Updating data	Stakeholders can be updated by editing the text

GDSE Step & Substep	Study Area: Key flows
Data purpose	<p>The GDSE distinguishes key flows, that are subsets of all possible waste/resource materials. Key flows help to reduce data size and look at specific flows in an isolated manner.</p> <p>Therefore, this substep requires a detailed description of the available key flows - their definitions, reasons for selecting specifically these key flows, any related supporting data sources, images, graphs or even videos.</p>
Data type	Formatted text, images, videos, links and/or tables
Data formats	Text or URL
Data upload procedure	Typed using user interface, images, videos and links are provided as URLs and therefore must be already available online beforehand
Special requirements	None
Updating data	Data on key flows can be updated by editing the text

GDSE Step & Substep	Status Quo: Flows
Data purpose	<p>Flows represent process data and dynamic relationships between actors inside and beyond the study area. Basically, flows in the GDSE represent material flows that have happened in a single year. Actors that participate in material flows can be either individual companies, or groups of actors (e.g households or</p>

	<p>small companies) in a geographical area.</p> <p>Flows can represent all types of material flows: from extraction sites to production, to consumption, to disposal and circular flows.</p>
Data type	Data tables and geospatial data
Data formats	TSV, CSV, XLSX
Data upload procedure	Detailed data upload and preparation procedure is available in the Appendix:
Special requirements	<p>The data should include flows that have happened in a single calendar year.</p> <p>Appendix lists all the additional special requirements to the dataset.</p>
Updating data	Updating flow data is explained in section 4.2 <i>Material Flow Data preparation and Data Entry</i>

GDSE Step & Substep	Status Quo: Flow Assessment
Data purpose	<p>Indicators are calculated on the fly by the system based on the provided data.</p> <p>However, inhabitant data supporting the calculation of indicators must be provided beforehand.</p>
Data type	<p>Indicators: user input</p> <p>Inhabitants: geospatial data</p>
Data formats	Inhabitants: CSV, TSV or XLSX with geometrical data provided as WKT
Data upload procedure	<p>Indicators are defined using the user interface controls.</p> <p>Number of inhabitants per administrative unit is uploaded together with the administrative units as explained in the Appendix, table Areas.</p>
Special requirements	Data on number of inhabitants should be from the same year as the other uploaded data, especially material flows.
Updating data	<p>Indicators can be updated through user interface</p> <p>Inhabitants can be updated by reuploading the Area table with the same unique area codes</p>

GDSE Step & Substep	Status Quo: Wastescapes
Data purpose	The module is in principle identical to the Study Area: Maps module, even though layers can be loaded here independently. The purpose of these map layers is to portray wastescapes in particular.
Data type	Geospatial data
Data formats	WMS Service
Data upload procedure	<p>Data is uploaded directly on GeoServer as explained in the section 4.4. <i>Integrating Geodata and Maps via WMS/WFS</i></p> <p>Maps can be grouped in categories & interactively rearranged in order by the interface users.</p> <p>The system also allows adding publicly available WMS/WFS layers that are useful as e.g. background maps.</p>
Special requirements	Maps need to be provided along with the layer style and a relevant legend.
Updating data	Maps can be updated by reuploading them on the GeoServer and again loading into the GDSE

GDSE Step & Substep	Status Quo: Description
Data purpose	Information on the Status quo can also be uploaded in a form of a report that can be consulted during the workshop. It can contain additional information, next to the maps and flows diagrams that are available in the Status Quo section.
Data type	Report
Data formats	PDF
Data upload procedure	<p>Uploaded using user interface</p> <p>Add report</p> <hr/> <p>Name</p> <input type="text"/> <p>PDF File</p> <p><input type="button" value="Choose file"/> No file chosen</p> <hr/> <p><input type="button" value="Cancel"/> <input type="button" value="OK"/></p>

Special requirements	None
Updating data	Reports can be updated by reuploading the files

GDSE Step & Substep	Status Quo: Objectives
Data purpose	<p>Listing all challenges and objectives that are related to the specific key flows or span the selected study area.</p> <p>During the workshop, the objectives will be ranked according to their importance to the group of stakeholders, converted into indicator values and evaluated after the simulation of a chosen strategy.</p>
Data type	Text
Data formats	-
Data upload procedure	Typed using user interface, cannot be uploaded in bulk
Special requirements	None
Updating data	Objectives can be updated by editing the text

GDSE Step & Substep	Targets: Ranking Objectives
Data purpose	Objectives are ranked during the workshop by the workshop participants from less to most important ones for the region (according to their opinion).
Data type	Order of objectives
Data formats	-
Data upload procedure	Workshop participants provide their ranking using the user interface guided by the workshop moderator
Special requirements	None
Updating data	Objectives are ranked during the workshop by the participants and shouldn't be updated after the workshop

GDSE Step & Substep	Targets: Flow Targets
Data purpose	Objectives from the <i>Status Quo: Objectives</i> substep are related with the indicators from the <i>Status Quo: Flow Indicators</i> substep by the workshop participants. In addition, they are required to set measurable targets for the indicator.
Data type	Relations, text and numbers
Data formats	-
Data upload procedure	Workshop participants enter the data using the user interface guided by the workshop moderator
Special requirements	None
Updating data	Flow Targets are set during the workshop by the participants and shouldn't be updated after the workshop

GDSE Step & Substep	Strategy: Solutions
Data purpose	The catalogue of the eco-innovative solutions that has been developed during the previous workshop steps is uploaded in the GDSE for the purpose of

	strategy building and simulation of changes.
Data type	Text and images
Data formats	Text, JPEG, PNG
Data upload procedure	Textual descriptions and supporting images are uploaded using the user interface
Special requirements	Each EIS requires 3 supporting images: 1) Diagram of the current process; 2) Diagram of the proposed process; 3) System diagram of the involved economic activities and flows between them
Updating data	Solutions can be updated by editing the texts and reuploading the images

GDSE Step & Substep	Strategy: Solution Logic
Data purpose	In order to be able to run numerical simulation of the solution effects on the system, solutions need to be converted into a set of rules in relation to the material flows.
Data type	Custom data structure
Data formats	-
Data upload procedure	Data is uploaded following the guidance of the user interface as explained in Deliverable 2.5, section 4.4. Strategy
Special requirements	Explained in Deliverable 2.5, section 4.4. Strategy
Updating data	Solution Logic can be updated through the user interface

GDSE Step & Substep	Strategy: Define Strategy
Data purpose	Strategies are created by the participants during the workshops and consist of multiple EIS implemented in chosen geographical areas. The strategies are later simulated to check how they would affect the current material flows and how the effect would look in the bigger picture and contribute to the earlier set targets.

Data type	Geospatial, text and numerical
Data formats	-
Data upload procedure	Workshops participants select EIS, answer parametric questions, draw polygons on the map and enter notes using the user interface guided by the workshop moderator
Special requirements	None
Updating data	Strategies are created during the workshop by the participants and shouldn't be updated after the workshop

GDSE Step & Substep	Strategy: Modified Flows & Flow Target Control
Data purpose	Both substeps provide simulation results in relation to previously entered data
Data type	Numerical data, can be exported into tables
Data formats	Relational database tables
Data upload procedure	Data is generated by building a graph and clicking "Calculate" button in the GDSE
Special requirements	-
Updating data	Data gets updated automatically after the user starts a new calculation for it's strategy; this does not affect calculations done by the other users

GDSE Step & Substep	Conclusions: Notepad
Data purpose	Organizing Consensus Levels & Organizing Sections are two groups of organizational headers that a PULL leader can use to analyze and expose the results of PULL process among the different participant groups after the last workshop
Data type	Text
Data formats	-

Data upload procedure	Typed using user interface, cannot be uploaded in bulk
Special requirements	None
Updating data	Notepad can be updated by editing the text through the user interface

GDSE Step & Substep	Conclusions: Sustainability
Data purpose	Sustainability report provides deeper sustainability-related insights into the EIS that have been generated outside of the GDSE
Data type	Report
Data formats	PDF
Data upload procedure	<p>Uploaded using user interface</p> <p>Add report</p> <div> <p>Name</p> <input type="text"/> </div> <div> <p>PDF File</p> <div> <input type="button" value="Choose file"/> No file chosen </div> </div> <div> <input type="button" value="Cancel"/> <input type="button" value="OK"/> </div>
Special requirements	None
Updating data	Reports can be updated by reuploading

GDSE Step & Substep	Conclusions: Workshop mode
Data purpose	All substeps of the Workshop mode bring the results and participant data from different participant groups into a single place and allow to explore the data using a series of visualisations
Data type	Textual, numerical and relational data, can be exported into tables
Data formats	Relational database tables

Data upload procedure	Data is automatically generated in the GDSE
Special requirements	-
Updating data	Data is automatically updated by the GDSE after any of the input data is changed by the users

GDSE Step & Substep	General: Focus Area
Data purpose	A smaller specific area that is not necessarily coincident with any official administrative units can be chosen for focused analysis. The chosen area appears throughout the GDSE process as a quick selection.
Data type	Geospatial
Data formats	GEOJSON
Data upload procedure	Uploaded by a data captain in <i>User Area -> Data Entry -> Bulk Upload -> Case Study Related</i>
Special requirements	Coordinates must be provided in WGS84
Updating data	Focus Area can be updated by copying a new geojson file or manually changing the coordinates

GDSE Step & Substep	General: Case study Region
Data purpose	Similar to a Focus Area, Case study Region is also a shortcut area that can be chosen for focused analysis. The chosen area appears throughout the GDSE process as a quick selection.
Data type	Geospatial
Data formats	GEOJSON
Data upload procedure	Uploaded by a data captain in <i>User Area -> Data Entry -> Bulk Upload -> Case Study Related</i>
Special requirements	Coordinates must be provided in WGS84, Region should include the Focus Area
Updating data	Case study Region can be updated by copying a new geojson file or manually changing the coordinates

4.2 Material Flow Data preparation and Data Entry

Material Flow data requires special treatment and preparation before it can be uploaded into the GDSE. The flow data is the basis for the simulations of effects caused by CE

strategies built during the PULL workshops and therefore needs to be prepared and entered into the GDSE following a predefined structure and semantics.

Material Flow data can be entered into the GDSE only by a Data Captain or an Admin. The entry happens through a Data Entry module which can be found in the User Area menu.

Bulk Upload

Tables of Material Flow data are uploaded in bulk through user interface. The GDSE then translates them into relational database tables. The structure and content of the required tables is explained in detail in the Appendix:

Each table has a specific template to be followed, templates can be downloaded by pressing a “template” button

After uploading a template, the log on the right will either confirm a successful upload or return back an error that prevented the data to be uploaded. In case of an error, no data from that table is uploaded. Errors that relate to the file structure (e.g. missing columns) are returned in the log, while errors related to data content (rows) also return an excel file that explains, in which rows an error has occurred. Only the first encountered error is returned even if there are other errors still remaining in the file.

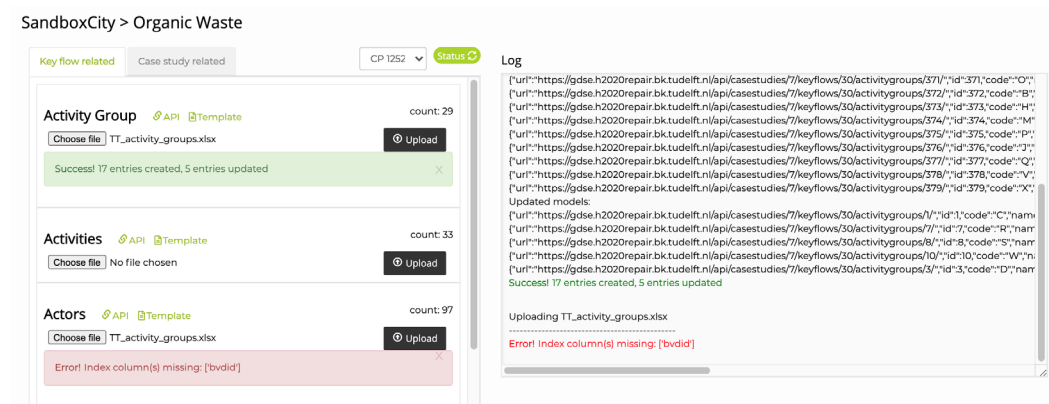


Figure 27: An example of a successful table upload (green) and an error (red).

Each table has a container with the following elements:

- API pointer which allows to see the uploaded data;
- Template download button
- File chooser & Upload button

- Counter of the total available data points in that table for the selected case study and key flow
- Status bar which appears after uploading and announces how many new data point instances have been created and how many were updated

Each table has one or multiple columns that represent a unique entry. The key is unique only for the specific combination of case study and key flow. Every time after a new table is uploaded, the script first of all checks, if a data point with that unique already exists, and if it does, updates the remaining attributes, otherwise creates a new instance. The key columns are indicated in the templates with a star sign (*). The status bar that appears after a successful upload, indicates how many entries have been updated and how many new instances have been created.

Edit Actors / Flows

It is possible to find and edit the uploaded data points and add new ones using a dedicated user interface. Initially meant as a data collection tool, later the user interface became deprecated due to the tedious work that would need to be done by a researcher.

Nevertheless, the detailed interface proves to be useful, when small adjustments need to be made to the data points. Additionally, it serves as a data overview and exploration tool that is useful for the identification of errors in the uploaded data.

Data Entry tab allows having an overview of all available activity groups, activities and actors. Selecting an actor allows exploring and editing all actor properties that are associated with it. In/Out Flows shows, which material flows and stocks the actor participates in and allows to edit any of the available fields.

Flow view tab is identical to the *Status Quo: Flows* subset accessible by anybody using the GDSE, however, allows quicker access to the filters and does not allow saving views. The tab allows querying the different data attributes and visualising them on an interactive Sankey diagram enhanced by a Sankey map.

Edit Materials

Every Material flow can be composed of one or multiple materials. However, the material itself can often be composed of multiple components (e.g. concrete is composed of sand and cement) and make parts of other materials or products. Especially in case of waste flows, material composition is often ambiguous and does not have strictly defined semantics and

level of detail. Therefore, a GDSE data captain is allowed to define their own material hierarchy that best fits the uploaded data.

Material hierarchy allows arranging materials in a tree-like structure, where each node can belong to only one other node and may or may not have siblings and may or may not have children. Material hierarchy can be edited by uploading a material table in Bulk Upload or by manually adjusting the names of materials in Material tree.

Nodes of the material tree that are already associated with a material flow cannot be deleted, however, their names can be edited.

The counter after each node shows, how many flows are associated with the material directly vs. how many flows are associated with the children of the material.

Material tree does not need to be balanced and is allowed to have more than one stem node.

SandboxCity > Organic Waste > Materials

▼ Material (directly used in flows / children in flows)
▶ Packaging materials (0 / 11 flows)
▼ Food products (0 / 19 flows)
Prepared meals and dishes (4 / 0 flows)
▶ Vegetable and animal oils and fats (0 / 5 flows)
▼ Processed and preserved fruit and vegetables (0 / 1 flows)
Other processed and preserved fruit and vegetables (1 / 0 flows)
Dairy products (4 / 0 flows)
▼ Prepared animal feeds (0 / 4 flows)
Prepared feeds for farm animals (4 / 0 flows)
▶ Grain mill products, starches and starch products (0 / 1 flows)
▶ Organic waste (0 / 62 flows)
▶ Beverages 2 (0 / 31 flows)

Figure 28: An example of a material hierarchy displayed in a tree-like structure

Delete AS-MFA data

Since no system is implemented to provide a quick and easy “undo” of data removal, deleting material flow data in bulk is not possible through the user interface. Only system admin is able to do that using an SQL query directly in the database. The SQL query is arranged into

separate SQL queries for each table. This the system admin can decide, which tables are cleared and which remain.

SQL queries allow clearing the data of only one selected key flow within one selected case study.

If any of the solutions and/or strategies have been created using the materials or activities already in the AS-MFA, those solutions and strategies will have to be deleted first using the same SQL query for deleting the data.

4.3 Exporting Data

The GDSE serves not only as a data visualisation platform, but also a platform to do data analysis, combine and filter datasets, and collect stakeholder input. Therefore, it allows exporting datasets that can later be used in different specialized data analysis software, e.g. GIS data analysis or LCA software. The data export is not targeted to any specific software and therefore provides the most system-agnostic data structure and data format.

Exporting data directly from the user interface

Portions of the material flow data can be exported using the user interface. All modules that allow displaying a Sankey diagram, allow the user to download the data behind the displayed Sankey. Data content can be adjusted by using the associated filters or views.



Figure 29: Red square indicates the button that allows downloading data which is visible in the displayed Sankey diagram.

The aggregation level of the data depends on the display level as explained in the table below.

Table 8: The aggregation level of the data depends on the display level

Display level	Data export columns							
	origin	origin_code	origin_wkt	destination	destination_code	destination_wkt	amount (t/year)	composition
Activity Group	Activity Group name	NACE letter	n/a	Activity Group	NACE letter	n/a	t	n/a
Activity	Activity name	NACE code	n/a	Activity name	NACE code	n/a	t	n/a
Actor	Actor name	Actor ID	Actor location	Actor name	Actor ID	Actor location	t	Flow composition

The same data export option is also available for the Sankey diagrams that display, how the simulated strategy affects material flows. The user can select to download status quo data, status of the modified flows or only the difference flows. In case of the difference compared to status quo, increased flows are displayed with a “+” sign, while decreased flows are displayed with a “-” sign.

Exporting data from the database

The rest of the data does not have a user interface for export and requires a targeted SQL query.

4.4 Integrating Geodata and Maps via WMS/WFS

The GDSE step ‘Study Area’ has a sub-step or screen called ‘Maps’ that shows spatial information about a particular study area. This sub-step features an interactive web mapping interface that shows spatial information as categorized web maps. Web maps consist of web mapping services (WMS/WFS) that can be case-study-specific (created, loaded and/or uploaded by Data Captains, and published using GeoServer) or existing external WMS or WFS map services (which are accessible through the URL of a website providing such services (e.g., OpenStreetMap, Leaflet, OpenLayers, Google Maps), even if these are password-protected.

To compose maps (i.e., arrange map layers) in this ‘Maps’ environment, the setup mode user can create map categories and add layers to a category by clicking on the “+ Layer” button (see figure 31). Next, a window appears prompting the user to select a map layer from a number of web map services and the GDSE’s dedicated GeoServer (which for REPAiR is accessible via this URL: <https://GeoServer.h2020repair.bk.tudelft.nl/>), which in turn contains PULL-case-specific map layers (Figure 30), and configure the import settings for this layer.

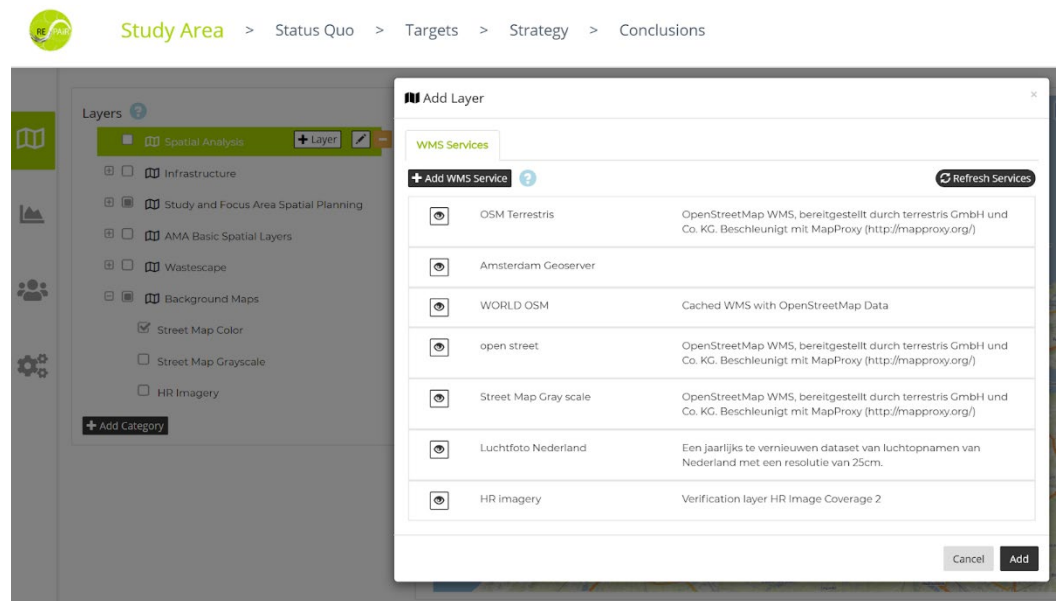


Figure 30: Adding a map layer to a category from the GDSE GeoServer.

Data Captains are responsible for uploading all the spatial layers with their appropriate styles on the GeoServer using the OSF workflow described in detail in the document available at this URL:

<https://mfr.osf.io/render?url=https://osf.io/gr762/?direct%26mode=render%26action=download%26mode=render>

The spatial data layers, which can be vector (e.g. ESRI SHP data format) or raster layers (e.g. GeoTIFF), must be first prepared before they can be uploaded onto the GDSE. The REPAiR Data Management Plan (DMP, page 13) shows guidelines on how to prepare SHP layers. Most important guidelines for SHP files are:

- A SHP layer must have a logical name, which clearly identifies what is being portrayed. See DMP for details on naming files (e.g., using “_” instead of spaces and using no special characters, et cetera).
- A SHP layer file name should be written using English language words.
- All SHP layers must share the same coordinate system: WGS84.
- Each SHP file must be accompanied by one style SLD file which specifies layer display and symbology for legends. SLD files can be generated using QGIS or ArcGIS.
- This SLD file should share the same layer file name and has .sld as an extension.

Uploading spatial data layers to the Open Science Framework (OSF)

REPAiR utilizes the OSF as the cloud repository for spatial data layers, and the main connection with the GeoServer web mapping platform, which is used to publish case-specific spatial information. For the case of REPAiR, spatial layers are uploaded to OSF, which places them on our REPAiR server in TU Delft. This way the layers are available for publishing on the GeoServer and that way available in the GDSE.

The OSF Data Platform and the GeoServer Workspace are linked through Nextcloud running as a cloud service on the TU Delft Server. The Nextcloud is integrated into OSF as an external storage. Data uploaded to the according folder in OSF are automatically transferred to the Nextcloud on the TU Delft Server. The GeoServer shares a common workspace with the Nextcloud, so that the uploaded shapefiles are accessible for the GeoServer. This is achieved by running Nextcloud and GeoServer as docker-containers which are started together by docker-compose.

OSF accounts must be created for each case study. Case-specific folders are created on the OSF platform, to hold spatial data layers that are uploaded.

Below the main steps for uploading spatial layers to OSF:

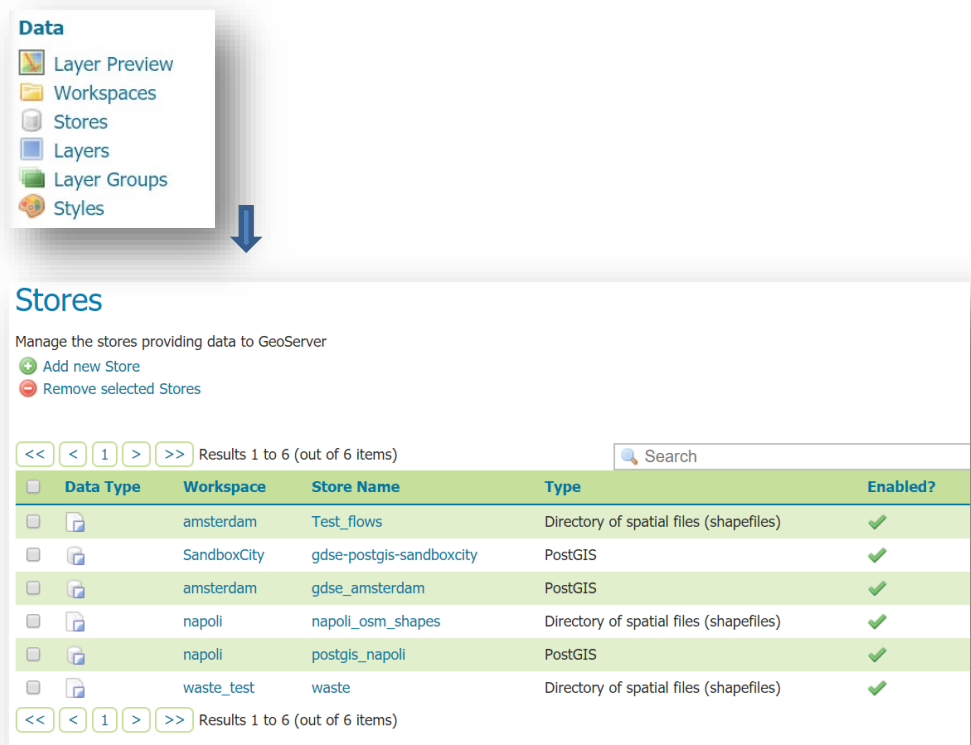
- Go to URL <https://osf.io> and log in with your credentials

- In your specific case study folder, find folder “Spatial_data” and storage provider ‘ownCloud’:
E.g. WasteREPAiR > Amsterdam > spatial_data > ownCloud: amsterdam
- Find a folder called T3.1_Study_Area, it has 2 subfolders: “shp” and “sld”
- “Shp” folder is for uploading all your shapefiles - please, do not create any subfolders, rather organise your files by giving them meaningful names (e.g. a keyword as the first word if they all belong to the same map)
- “Sld” folder is for uploading all your style files, do not create any subfolders there either.

Publishing a spatial layer in REPAiR's GeoServer

Once the files are uploaded into the TU Delft server they need to be published in the GeoServer. Users must have GeoServer credentials to be able to publish uploaded spatial data layers.

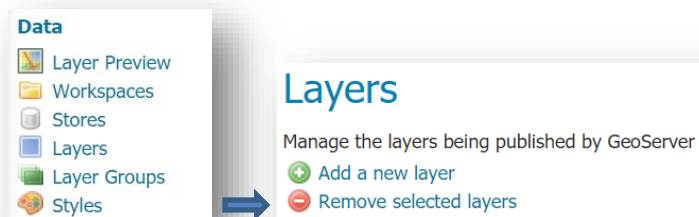
- Go to URL <https://GeoServer.h2020repair.bk.tudelft.nl/GeoServer/web/>
- Sign in with your GeoServer credentials
- <Uploaded spatial layers are now available for fetching here. Find a case study by clicking on 'Stores' on the left hand side panel. Type must be 'Directory of spatial files (shapefiles)'. You will upload and publish new layers to this store.



The screenshot shows the GeoServer web interface. On the left, a 'Data' menu is open, listing options: Layer Preview, Workspaces, Stores, Layers, Layer Groups, and Styles. A blue arrow points from the 'Stores' option to the main 'Stores' page. The 'Stores' page has the title 'Stores' and a subtitle 'Manage the stores providing data to GeoServer'. It includes two buttons: 'Add new Store' (with a green plus icon) and 'Remove selected Stores' (with a red minus icon). Below these is a table with 6 items, showing pagination 'Results 1 to 6 (out of 6 items)' and a search bar. The table columns are 'Data Type', 'Workspace', 'Store Name', 'Type', and 'Enabled?'. All items are enabled, indicated by green checkmarks.

Data Type	Workspace	Store Name	Type	Enabled?
	amsterdam	Test_flows	Directory of spatial files (shapefiles)	✓
	SandboxCity	gdse-postgis-sandboxcity	PostGIS	✓
	amsterdam	gdse_amsterdam	PostGIS	✓
	napoli	napoli_osm_shapes	Directory of spatial files (shapefiles)	✓
	napoli	postgis_napoli	PostGIS	✓
	waste_test	waste	Directory of spatial files (shapefiles)	✓

- On the left-hand side panel, click on 'Layers' to view all available layers for publishing. Next, then click on 'Add a new layer':



- On 'Add layer from', a drop-down menu shows available stores, choose an adequate destination store.
- Select the layer to be published. Click on 'Publish'.


Published	Layer name	Action
✓	AFH01_Periurban	Publish again
	AFH01_Focus%20Area	Publish

- Edit the layer attributes where required. Basic Resource Info, Keywords, Metadata links, Coordinate Reference Systems, Bounding Boxes.
- Input is required on 'Bounding Boxes'.
On 'Native Bounding Box', choose 'Compute from data'
On 'Lat/Lon Bounding Box', choose 'Compute from native bounds'

Bounding Boxes

Native Bounding Box


Min X	Min Y	Max X	Max Y
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

[Compute from data](#) 

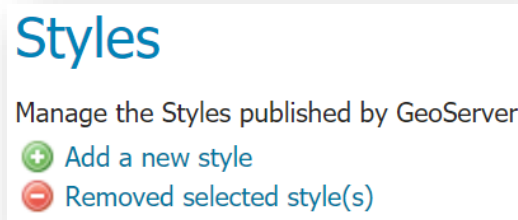
[Compute from SRS bounds](#)

Lat/Lon Bounding Box

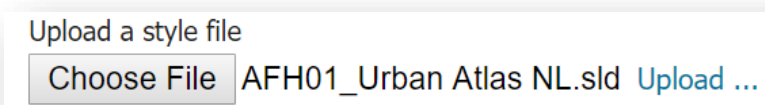
Min X	Min Y	Max X	Max Y
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

[Compute from native bounds](#) 

- Now the layer is published but it still does not have a style associated with it. On the left panel, click 'Styles' to see all styles available. We want to upload a style which is associated with an SLD file. Click on 'Add a new style'. Choose Workspace 'Amsterdam'.

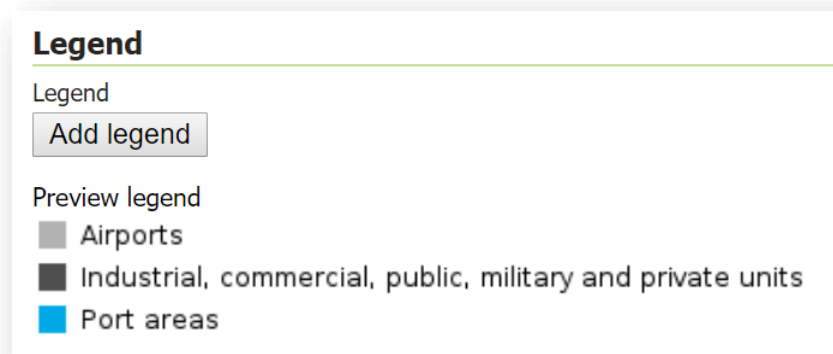


- On 'Style Content' click on 'Choose File' and navigate to the target SLD. Click on 'Upload'.



'Name of Style' and 'Style Editor' get automatically filled out.

- Under 'Legend', click on 'Preview legend' to preview the style's legend. The legend should appear right under 'Legend'. Click 'Apply' to save changes to the style.



- Now the new style needs to be associated with the right layer. Click on tab 'Publishing' to associate the style with a previously published layer. You will see a list of published layers. Select the target layer by checking options 'Default' and 'Associated' on the target layer.

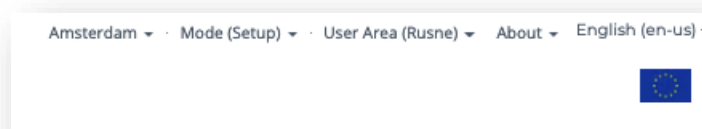
Workspace	Layer	Default	Associated
napoli	clc12	<input type="checkbox"/>	<input type="checkbox"/>
napoli	osm_buildings_a_free_1	<input type="checkbox"/>	<input type="checkbox"/>
napoli	osm_railways_free_1	<input type="checkbox"/>	<input type="checkbox"/>
SandboxCity	OperationalLocationsSandbox	<input type="checkbox"/>	<input type="checkbox"/>
SandboxCity	SandboxAdministrativeLocations	<input type="checkbox"/>	<input type="checkbox"/>
SandboxCity	asmfa_administrativelocation	<input type="checkbox"/>	<input type="checkbox"/>
amsterdam	AFH01_Perurban	<input type="checkbox"/>	<input type="checkbox"/>
amsterdam	AFH01_Urban_20Atlas_20NL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
amsterdam	Focus Area	<input type="checkbox"/>	<input type="checkbox"/>
amsterdam	Studyarea	<input type="checkbox"/>	<input type="checkbox"/>
amsterdam	test_flows	<input type="checkbox"/>	<input type="checkbox"/>

- Click 'Apply' to save. Tab 'Layer Preview' allows you to visualize the layer with the associated style. If nothing appears on this tab, make sure you allow all scripts on this site in this browser's tab.
- Click 'Submit' to complete the upload. The new style will appear on available styles in the given Store.

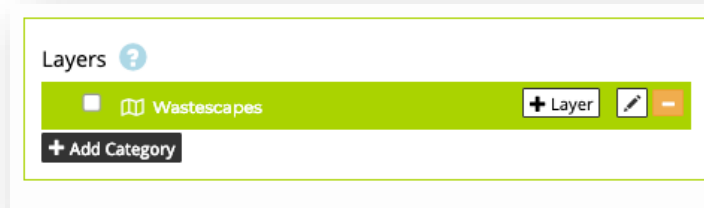
Loading layers in the GDSE

Now all your layers are available for display in the GDSE. Below the steps to compose maps using the recently published spatial data layers, and the desired existing web services.

- In the top right corner choose "Setup" mode and in the top left menu "Study Area" step



- Click on "Add Category" to create a new category for your maps (one category consists of multiple layers, it is up to you how to organise them)
- When you select a category a button +Layer will appear for adding spatial layers to it



- Select layers that you want to add
- You can reorder the layers by dragging and dropping and selecting the checkboxes for their inclusion in the Workshop mode

After adding or changing WMS-Services, the “Refresh Services” button must be pressed in the setup mode to update the list of available layers, as shown in the screenshot of Figure 31 below.

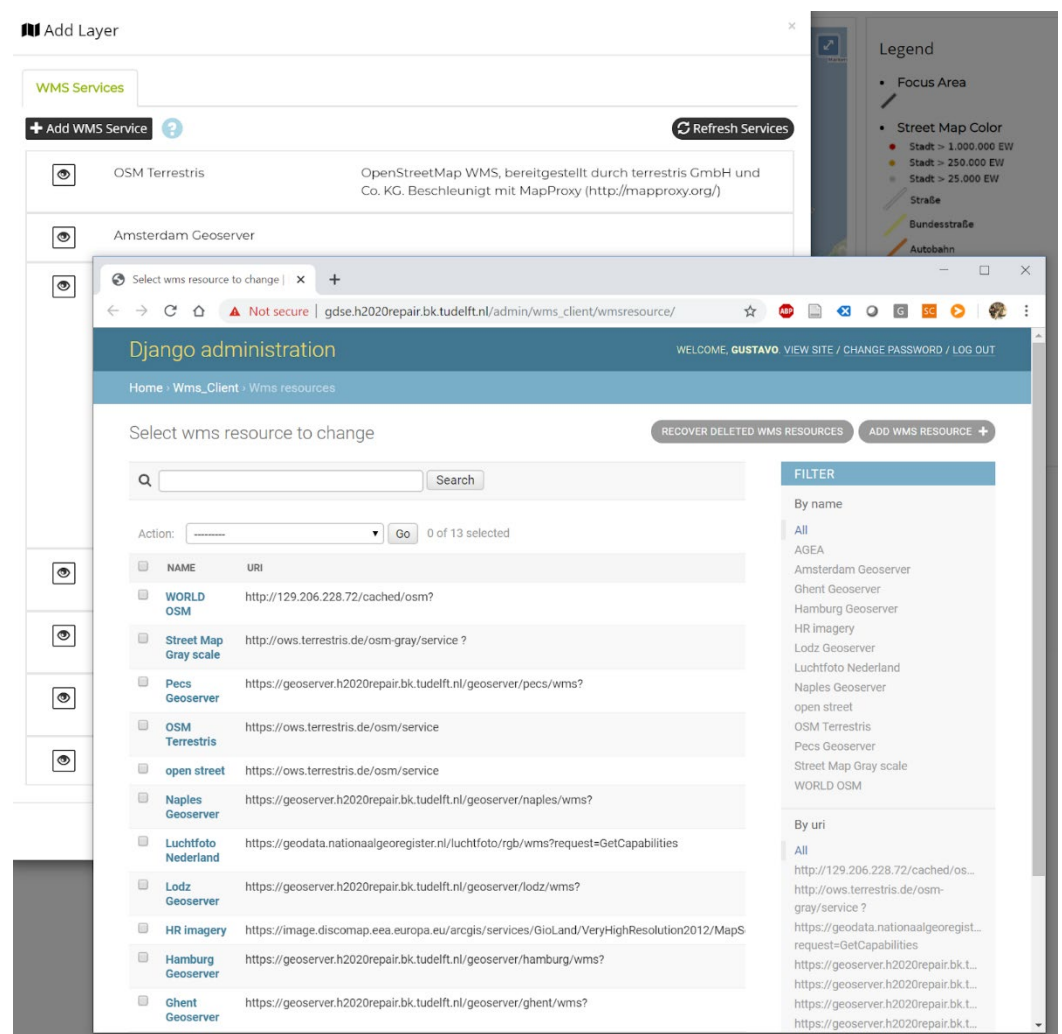


Figure 31: Adding an external WMS or WFS layer service in the Study Area step.

4.5 Open Data Policy and Restrictions: User Management and Access Rights

The REPAiR project has been part of the H2020 Open Science Pilot which requires all project data and results to be “as open as possible, as closed as necessary”. However, data from the waste management companies and governmental waste divisions is considered sensitive and confidential data and therefore got entrusted to the REPAiR consortium only under strict confidentiality agreements. To be able to comply with the confidentiality agreements and avoid accidental data leakage, the GDSE does not have public access and every GDSE user is required to login using a username and password.

There are five groups of users available that are allowed to access different modules of GDSE. The table below gives an overview of the User roles and available groups of permissions.

In addition to the assigned roles some users may have a “Superuser status” which means that they are admin users. This status designates that the user has all permissions without explicitly assigning them.

Table 9: User Management and Access Rights

Permission	Admin	Data Captain	Pull Leader	Researcher	Workshop Participant
Create users, assign case studies and user rights	✓	✓			
Create and delete case studies	✓				
Add key flows	✓	✓			
Access Admin Area	✓	✓			
Access Setup Mode	✓	✓	✓		
Access Workshop Mode	✓	✓	✓	✓	✓
Modify solutions	✓	✓			
Create strategies	✓	✓	✓		✓
View conclusions	✓	✓	✓		
Add notes to conclusions	✓	✓	✓		

Every user is granted access to one or more case studies and has the listed permissions only for those designated case studies. Only superusers are allowed to assign case studies to the other users. This way it is possible to control who is able to view the sensitive data.

User's responsibility

Users who are granted the access take the responsibility for keeping their password safe and not sharing it with the other people within or outside of the consortium as GDSE does not collect any user statistics.

User rights can be added and removed by the admin user who has access to the *Admin Area*.

Workshop mode

Additional level of data confidentiality protection is created by allowing users that can access setup mode to choose to anonymize actors of the material flows while the chosen views are displayed in the workshop mode. The choice can be made per created view by selecting the ☐ visible in workshop mode checkbox. If the checkbox is selected, the view in workshop mode anonymizes all actors for all users both in Status Quo as in the Modified Flows module.

5 Outlook of possible further development

Refactoring

There still remain relics of early stages of the development in the code that should be removed for the sake of clarity. These include the Activity2Activity and Group2Group models that are not used in production anymore.

The FractionFlows mentioned in chapter [Data Structure](#) de facto replace the Actor2Actor and ActorStock models. Removing the old structure that is still in use in parallel is expected to be a rather laborious task, concerning the bulk upload, the API routes and several models. Changing the models might also have a heavy impact on existing data in the database.

Rest API

The triggers to build base and strategy graphs are posted to the API. The frontend waits for the response to this request to find out when the backend has finished building the calculations. As this can take a long time, timeouts might occur in the frontend leading to confusing error messages. There also is no detailed information about the status of the calculations available via the API.

A better solution would be to post the request without waiting for a response and to acquire status information over a socket connection between frontend and backend. In addition, a logger should be implemented in the backend keeping track of the progress of the calculations.

There is an option to anonymize the actor names in the backend. However, this is not a real anonymization. The backend still serves the names but they are hidden in the frontend. An experienced user could read the names from the network traffic in the developer tools in the browser. Real anonymization in the backend is advised for future development.

Performance

The filtering of flows and the strategy calculation including the Graph Walker algorithm are the most time-consuming operations in the backend. They are already fairly optimized but there is still a lot of potential for optimization.

The flow map and the interaction with the Sankey in the frontend also could be optimized. When clicking the “All” button to select all flows in the Sankey, every single flow is requested in an individual request to the server. The backend filter route already provides faster ways to request all at once.

The clustering of actors on the map is done frontend-side. To increase the performance, the clustering could be implemented in the backend.

Texts

All of the error messages, that pop up in the frontend, are generated in the backend. Not all errors in the backend are caught, resulting in occasional “500 Internal Server Error” messages without detailed information attached.

Furthermore, many messages from the backend are very generic and not necessarily understandable or relevant to the common user. Some could already be avoided by checking the user inputs more frequently in the frontend, but are not implemented yet due to a lack of time in the development.

The current implementation of the translations was not made with keeping all possible languages in mind. Instead of translating whole sentences with variables marked inside, sentences are often disassembled into their parts, placing variables in between. This leads to a heavy fragmentation in the translation files and the reconstruction of the sentences might not be applicable to all languages.

More flexible indicator definitions

Indicators can be calculated at the moment related to the total number of inhabitants in an area. The population data has to be provided beforehand and uploaded to the GDSE. For certain questions, a more generic definition of indicators could be required, relating flows in a certain area to certain household or company types in that area. A technical solution could be to define the indicators based upon WFS-Layers, which deliver the required data for the area as points or polygons and could be defined very flexible for a project.

Integration of external APIs

In addition, WFS-Layers could be used to calculate additional sustainability indicators like the accessibility to facilities or the number of inhabitants exposed to certain emission levels. The calculation (Accessibility Isochrones, Odour Contours etc.) could be calculated using external APIs, the GDSE would be used to integrate the results and calculate the relevant indicators and visualize the results.

Follow-up repositories

Up to date, there are already a couple of repositories that have been forked from the REPAiR project for different purposes:

- CINDERELA is an H2020 project that has used the Status Quo module of the GDSE for the analysis of material flows.
<https://github.com/H2020Cinderela/Cinderela-Web>
- Amsterdam Nulmeting has been a follow-up project by the TU Delft in collaboration with the municipality of Amsterdam that provided a material flow analysis for the Amsterdam Circular Economy Strategy 2020-2025. The analysis has been based on the advanced Material Flow modules of the GDSE.
<https://github.com/VasileiosBouzas/geoFluxus>
- The material flow analysis module has been further rebuilt to include a series of additional visualisations and to couple material flow data with the carbon emissions as part of the CINDERELA h2020 project.
<https://github.com/VasileiosBouzas/geoflux>
- geoFluxus is a spin-off company that has been started by the REPAiR project members in 2020 and continues developing the GDSE and implementing the tool as an open source Circular Economy monitor for governments and corporations.

Repository: <https://github.com/geoFluxus/geofluxusApp>

More information about the project can be found here: <http://geofluxus.com/>

Appendix

This document has been used throughout the REPAiR project as a handout to the data captains who have been preparing the material flow data for the upload into the GDSE.

It has been the responsibility of the data captains to prepare the data according to the templates using their original datasets.

Material Flow Data Upload Instructions

All datasets need to be uploaded on OSF by the data captains.

Each dataset must be uploaded along with the metadata.txt

All datasets should be uploaded following the order of the timeline below:

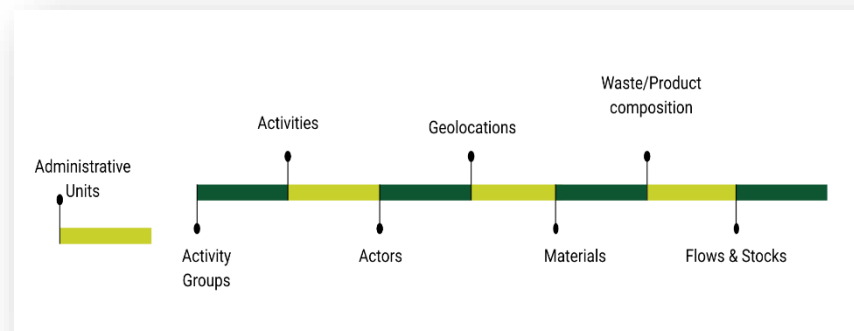


Figure 32: Order of upload for different data sets

If needed, the datasets can always be updated later during the project. However, it is important that each dataset that appears later in the timeline is dependent on all of the previous datasets and therefore cannot be uploaded earlier than them.

Data Preparation Process

The basic steps to convert data from its original format into the GDSE tables are as following:

1) Match actors in your data with the actors in the GDSE database.

There are going to be two types of actors in your data: companies (waste producers, waste collectors, waste treatment facilities) and households.

For the **companies**, you can match them to the ones from the ORBIS database by the combination of postcode + street name + house number, which should be unique for each address. It is not always a correct match but mostly it works and at least this way we get the geographical location right even if multiple companies are registered under the same address.

However, if your data is already linked with NACE codes, then you should also check that matching it with a company in ORBIS should not change the original NACE code.

You can find the ORBIS list of all companies that have been extracted for your case study on OSF folder "T3.2_Actors_--keyflow--".

If the companies are not to be found in the ORBIS database (or simply cannot be matched because of the differences in names & addresses), you need to add them as extra actors. This means that you need to prepare a .tsv file "T3.2_Actors_extra" (see section "Actors" of this document for detailed explanations on how this table needs to be filled) and upload it into the same folder "T3.2_Actors_--keyflow--" (with the metadata!). The companies need to be given a new identifier instead of the BvDid (however still in the column with the same name). The identifier can be any combination of numbers and letters as long as it is unique (e.g. GENT000023). You should have this one file of the extra actors consistent throughout all of your data: if you encounter these same companies in the other datasets, they should not get a new identifier but the one that has been already created.

If you have a list of companies with their names, it is also possible to run an ORBIS search in their database based on names - this increases the chances of a correct match. If you would like this to be done for you, you should contact somebody from the TU Delft team to run it for you (e.g. Rusne or Alex)

If companies do not have an address, there is no error but if there is no address, there will be no point on the map and that flow will simply not appear in the GDSE maps.

For the **households** you need to match them to the administrative codes that have been provided together with the shapefiles some time ago by the data captains - check your

files on the OSF to see what unique identifiers were provided. These need to be provided again as origins or destinations in the Flow & Stock tables (see the Tables and Explanations section for detailed explanations).

A single actor is:

1. only in one location (address);
2. has only one name;
3. has only one role in a waste chain, i.e. if the same company is waste producer (first one in the flow chain) and waste treatment (last one in the chain), then it needs to be split into multiple actors with separate unique identifiers (name and location can stay the same)

2) Prepare the Material table.

The material table can be prepared in two ways - by providing a .tsv file with all the materials (see section Materials for the detailed explanations) or by using the GDSE interface on <https://gdse.h2020repair.bk.tudelft.nl/data-entry/> at the section "Edit Materials". The materials can be added, deleted or renamed where needed.

3) Prepare the Composition table.

Composition table mainly shows how much of which material is in which flow amount. These materials must match with the materials in your material hierarchy (the previous step) (see the Waste / Product Composition section for a more detailed explanation)

4) Prepare the Flows & Stocks.

Flows are the most important part of the AS-MFA framework as they demonstrate how the materials are flowing from one actor to another, whether the actor is a set of households or an individual company.

Stocks demonstrate where materials accumulate, i.e. where they stay for longer than a year. The only difference between the Flow and Stock tables is that Stocks do not have destination, which means that materials are kept at the location of the actor of origin.

See the Flows & Stocks section for a more detailed explanation.

5) Upload files on the OSF.

The final files (tables) that need to be provided are:

T3.2_Actors.tsv with the list of additional actors that were not in ORBIS. I suggest to keep this list in one single file for all the data you have in order to avoid duplicates and for the remaining data always double-check both ORBIS identifier (BvDid) and this list.

T3.2_Materials.tsv is optional as material hierarchy can also be created manually using the GDSE interface.

T3.2_Composition.tsv - can be split into multiple files for each type of waste or can be one file - however it is more convenient to you.

T3.2_Flows.tsv - it is convenient to split this according to the different actor to actor relationships but not necessary from the GDSE point of view.

Tables and Explanations

Activity Groups			
Code	Name		
G	Wholesale and retail trade		
E	Waste management and remediation		

Activities		
NACE	Name	AG
G-4724	Retail sale of bread	G
E-3811	Collection of non-hazardous waste	E

Actors			
BvDid	Company name	...	NACE
NL0007543	Bakery A	...	4724
LMA00012	Waste Company		3811

Materials		
Level I	Level II	Level III
Farinaceous products		
	Bakery	
		Bread
Plastic		
	PET plastic	

Sources			
Author	Year	Title	Citekey
CREM	2016	Bepaling voedselverspilling in huishoudelijk afval Nederland 2016	crem2016
LMA	2018	Landelijk Meldpunt Afvalstoffen	lma2018

Compositions					
NACE	Custom name	Material	Fraction	Avoidable	Source
A-4724	02212 Packaged bread	Bread	0.9	TRUE	crem2016
		PET plastic	0.1	FALSE	crem2016
A-4724	023214 Plastic packages	PET plastic	1.0	FALSE	lma2016

Flows						
Origin	Destination	Amount	Composition	Year	Waste	Source
NL0007543	LMA00012	10	02212 Packaged bread	2016	TRUE	lma2016

Stocks						
Origin	Amount	Composition	Year	Waste	Source	...
NL0007543	100	023214 Plastic packages	2016	FALSE	lma2016	...

Figure 33: Relationship between all tables

The relationship between all tables that need to be delivered for the AS-MFA input into the GDSE. Those cells which are fully filled with a colour show which table the primary key comes from. When the key reappears in a different table, it is highlighted with the same colour. E.g. NACE codes in the tables “Actors” and “Compositions” must be present in the table “Activities”.

Administrative Units

Note: Administrative units are related to a case study and not a key flow, therefore the same administrative units will be used for all key flows of the same case study.

Area Levels

Unique Key: [Name; Level]

Table 10: Template of “Area Levels”

Name	Level
<i>Name of the administrative unit, ideally from the list below, can also be given in a local language</i>	<i>Corresponding level of the administrative unit, ideally according to the list below</i>
Comune	8

- 3-4 levels of administrative units need to be chosen per case study, according to the granularity of available data, size of the focus area, governance structure.
- Each level must correspond to one of the levels in the table below. Matching terminology is not important, however the hierarchy must match. I.e. it can happen that in one country “Municipalities” are composed of “Districts”. In that case “Districts” can be called “CityDistricts” as they are lower down the hierarchy chain than municipalities. Also if “Municipalities” are the same as “NUTS3”, they can be assigned to whichever level as long as the overall hierarchy between all the units is consistent (administrative units higher up the hierarchy must consist of the ones lower down the hierarchy and not the other way round).
- It is better to use internationally accepted units, e.g. NUTS and geometry provided by them for the overall consistency.

1	World
2	Continent
3	Country
4	NUTS1
5	NUTS2
6	NUTS3
7	District
8	Municipality
9	CityDistrict
10	CityNeighbourhood
11	CityBlock
12	StreetSection
13	House

Areas

Unique Key: [Code]

Table 11: Template of “Areas”

Parent	Level	Inhabitants	Name	Code	WKT
<i>Code of the higher level administrative boundary (if applicable)</i>	<i>Level number of the administrative unit as indicated in the “Area Levels” table</i>	<i>Number of inhabitants in the area</i>	<i>Name of the area</i>	<i>Unique administrative code of the area, must be unique also across all administrative units</i>	<i>WKT geometry of the area in WGS84</i>
NL	4	325 678	Noord Holland	NL001	POLYGON((-71.17765850 52917 42.39029097 39571,...))

- Units have to be topologically consistent. A lower level unit cannot belong to multiple higher level units, e.g. one municipality cannot belong to multiple districts.

- Not all units must have corresponding lower level units provided. E.g. If the whole country is partitioned into provinces and the provinces are partitioned into regions, then regions *can also be* supplied for one of the relevant provinces only and not for the whole country.
- Geometries must be provided in WGS84 EPSG:4326. Polygon geometries must be topologically valid.

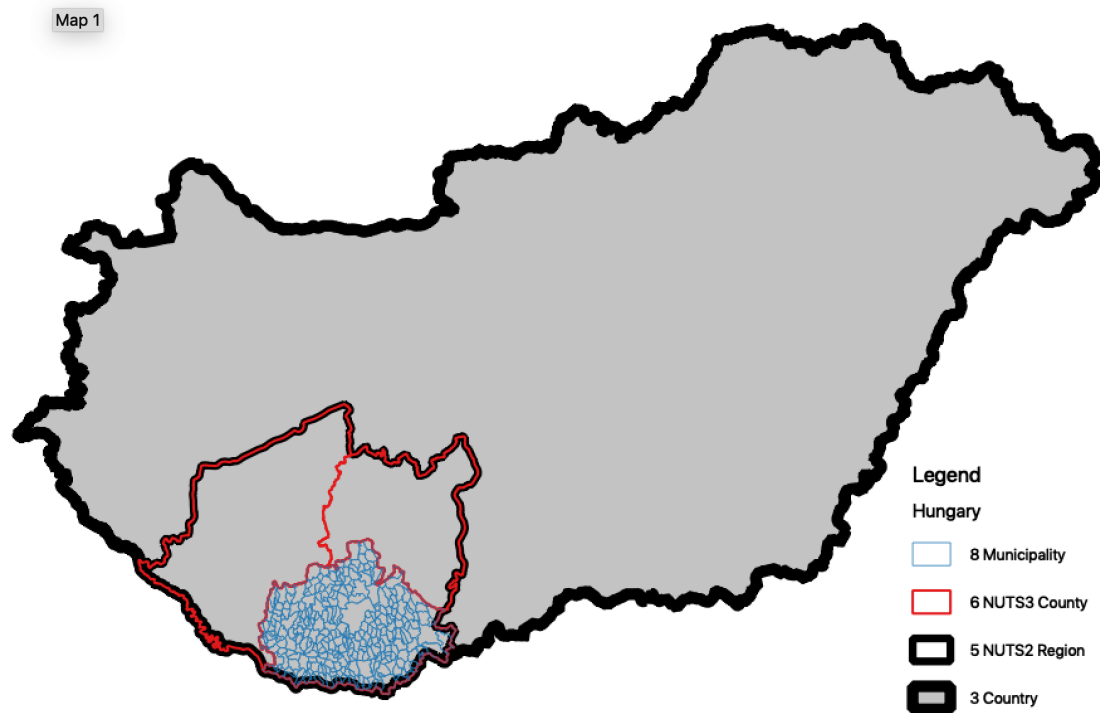


Figure 34: Example of the Administrative Units for the Pecs case study

Activity Groups

Unique Key: [Code]

Table 12: Template of "Activity Groups"

Code	Name
<i>A*38 code that represents Activity Group</i>	<i>Name of the Activity Group as in NACE Rev. 2</i>
A	Agriculture, Forestry and Fishing

Additional dummy codes can be added if necessary:

- 1) **V** Consumption in households
- 2) **WE** Export
- 3) **WI** Import
- 4) **WU** Unknown

They belong to division **00**.

Activities

Unique Key: [NACE]

Table 13: Template of "Activities"

NACE	Name	AG
<i>4 digit NACE code in format "A-0111"</i>	<i>Name of the Activity as in NACE Rev. 2</i>	<i>A*38 code that represents Activity Group. Must be present in the list of Activity Groups</i>
A-0112	Growing of rice	A

List of all NACE activities and their correspondence with the activity groups:

<https://osf.io/vb8jr/>

We are using only the 4 digit NACE codes.

Additional dummy codes:

- 1) **V-0000** Consumption in households
- 2) **WE-0001** Export
- 3) **WI-0002** Import
- 4) **WU-0003** Unknown

Actors

Unique Key: [BvDid]

Table 14: Template of "Actors"

BvDid	name	NACE	code	year
<i>Identification used by ORBIS. Can also be any other custom unique identifier.</i>	<i>Name of the company</i>	<i>4 digits or letter + 4 digits, must be present in the list of Activities</i>	<i>Consolidation code according to ORBIS (optional)</i>	<i>Last time the company has been reported</i>
NL000786543	Orgaworld B.V.	E-3623	C1	2016

description english	description original	BvDii	Website	employees	turnover
<i>Description (optional)</i>	<i>(optional)</i>	<i>ORBIS BvD Independence Indicator (optional)</i>	<i>(optional)</i>	<i>(optional)</i>	<i>thousands in EUR (optional)</i>
		U	www.orgaworld.nl	298	1200

The template is based on the extracts of the ORBIS database, however, any additional necessary actors can be added later either manually one by one using a GDSE interface or by uploading a bulk table following the same template.

The need for additional actors can arise from the available data if:

- it includes companies that have not been registered in the ORBIS database;
- It includes companies that are outside of the chosen study area and therefore were not included in ORBIS extracts, however still participate in the relevant flows (e.g. a significant part of waste generated in the study area can be treated in a waste treatment facility which lies in another province or even a neighbouring country)
- It includes companies that perform economic activities which have not been included in the original list of the anticipated economic activities (e.g. food waste may be generated by companies that are not related with food production or services, e.g. educational institutions, hospitals, etc.)
- companies cannot be matched to the ones registered in ORBIS database as they have different addresses.

All actors need to belong to one of the activities in the “Activities” table. If an activity of an actor is (temporarily) unknown, a new activity can be added based on the “Activity Groups” by adding (unknown) to the Activity Group name, e.g. “E-0099 Waste Management (Unknown)”. These actors can be later reclassified to match the actual economic activities they are carrying out.

Not all actors need to be companies. For now the following non-company actors are supported in the GDSE database structure, although additional ones can be created upon request:

- **Households.** Households are involved in the NACE activity “V-0000 Consumption in Households”. They need to be matched with one of the Administrative Units that the data is aggregated to. The GDSE also supports providing data on a single house level if such kind of data is available, however it is far more likely to have aggregated data on neighbourhood, district or even municipality level. In that case an administrative units becomes an actor and needs to be input into the GDSE following the same template, as in the example below.

Table 15: Example of household type actors

BvDid	name	NACE	code	year
-------	------	------	------	------

WK003403	Almere Buiten	V-0000		
WK036200	Amstelveen	V-0000		

description english	description original	BvDii	Website	employees	turnover

- **Export /Import.** If it is known which country/continent the certain flows go to or come from, however, a specific actor (company) is not known, it is possible to add flows towards generic chosen areas which will then belong to the activities “WE-0001 Export” or “WI-0002 Import”

Actors can be edited manually one by one using the GDSE interface as in the image below.

AKKERBOUW- EN PLUIMVEEBEDRIJF KNOOK- DE VISSER V.O.F.

Upload Changes

Included ☒

Name **AKKERBOUW- EN PLUIM**

Activity **A-0111 Growing of cereals (except rice), leguminous crops and oil seeds**

Website

Year **2017**

Turnover

Employees **2**

BvDId **NL32151033**

BvDii **U**

ConsCode **LF**

Description **Teelt van aardappels, suikerbieten en overige wortel- en knolgewassen. Teelt van granen, peulvruchten, oliehoudende zaden. Houden van pluimvee.**

Locations

Administrative Location

Remove	Name	Marker	Area	Edit
			(5.54, 52.42)	

Operational Locations

Remove	Name	Marker	Area	Edit
+ Add				

Figure 35: Example of an actor in the GDSE interface that can be found under “Data Entry” → “Edit Actors”

Actor Locations

Unique Key: [BvDid]

Table 16: Template of “Geolocated Actors”

BvDid	Postcode	Address	City	Country	wkt
<i>Identification used by ORBIS. Can also be any other custom unique identifier (must be present in the list of Actors)</i>	<i>(optional)</i>	<i>(optional)</i>	<i>(optional)</i>	<i>(optional)</i>	<i>in WGS84, at least 6 digits after comma</i>
NL000786543	1876 JK	Meerlanders traat 6	Amsterdam	Netherlands	POINT(4.987659 54.987654)

Some of the companies come geolocated in the ORBIS database, in that case the longitude/latitude fields can be used to find the exact locations. The locations are provided with the precision of decimal second which may result into positional bias of up to 111.32m depending on the relative position to the equator.

Actors that do not have their locations provided by the ORBIS database or those that have been added later, can be located using the geolocation service. It is advised to use the BatchGeo service for consistency throughout the project.

Geolocation can also be entered, edited or updated using the “Actor Edit” tab in the GDSE “Data Entry”.

More detailed instructions on how to perform geolocating can be found here: https://docs.google.com/document/d/1DljCio2LAjr2er_mJvM4OSjdvsHbaAOAttOCdeCvSzM/edit

Materials

Unique Key: [Name]

Table 17: Example of the Materials hierarchy and its corresponding table for the GDSE

Parent	Name
<i>Material parent in the hierarchy, leave the field empty if the material belongs to the level I of the hierarchy</i>	<i>Material name</i>
Cereals	Wheat

Level I	Level II	Level III	Level IV	Level V	Parent	Name
Products of agriculture, hunting and related services					Products of agriculture, hunting and related services	Products of agriculture, hunting and related services
	Non-perennial crops				Products of agriculture, hunting and related services	Non-perennial crops
		Cereals			Non-perennial crops	Cereals
			Wheat		Cereals	Wheat
				Durum wheat	Wheat	Durum wheat
				Wheat, except durum wheat	Wheat	Wheat, except durum wheat
			Other oil seeds		Cereals	Other oil seeds
				Sunflower seed	Other oil seeds	Sunflower seed
			Seeds for forage plants		Cereals	Seeds for forage plants
				Beet seeds	Seeds for forage plants	Beet seeds
				Raw vegetable materials	Seeds for forage plants	Raw vegetable materials
	Perennial crops				Products of agriculture, hunting and related services	Perennial crops
		Grapes			Perennial crops	Grapes
			Table grapes		Grapes	Table grapes
Packaging materials					Packaging materials	Packaging materials
	Board				Board	Board
		Corrugated boxes			Board	Corrugated boxes
		Folding boxes			Board	Folding boxes
		Cartons/plates/cups			Board	Cartons/plates/cups
		Miscellaneous board			Board	Miscellaneous board
			Beverage cartons		Miscellaneous board	Beverage cartons
			Paper plates & cups		Miscellaneous board	Paper plates & cups
			Cards & labels		Miscellaneous board	Cards & labels
			Egg boxes & alike		Miscellaneous board	Egg boxes & alike
			Other board		Miscellaneous board	Other board

▶ Food products (read only)
▶ Beverages (read only)
▼ Products of agriculture, hunting and related services (read only)
▼ Non-perennial crops (read only)
▼ Cereals (except rice), leguminous crops and oil seeds (read only)
▼ Wheat
Durum wheat
Wheat, except durum wheat
▼ Other oil seeds
Other oil seeds n.e.c.
Sunflower seed
Sesame seed
Rape or colza seed
Mustard seed
Lin seed
▶ Soya beans and groundnuts
▶ Dried leguminous vegetables
▶ Maize

Figure 36: Example of the "Edit Materials" tab in the GDSE "Data Entry" interface

Waste / Product Composition

Unique Key: [Name]

Table 18: Template of the "Composition" table

NACE	Name	Material	Fraction	Hazardous	Avoidable	Source
A-0001, must be present in the list of Activities	Unique name for the composition. If possible, should contain EWC code	Composing material (must be present in the Material list)	Part of the corresponding material. All fractions in one composition must add up to 1	TRUE or FALSE, also FALSE if not applicable	TRUE or FALSE, also FALSE if not applicable	BibTEX key for the source (must be present in the list of publications)
A-0112	02212 Roofing materials	Wood	0.7	FALSE	FALSE	lma2016
		Steel	0.3	FALSE	FALSE	lma2016

Composition table is needed to clarify what exactly every single flow consists of. As data can refer to the different compositions containing the same material, it is important to split the flow into its fractions as much as the data allows to do so. An easy example of the flow composition would be waste flow from a bakery which throws away 1 tonne of packed bread. If it is known that bread is packed in a plastic package which weighs 50g for each 1kg of bread, then the composition of such product (or waste) is:

$$0,05 / (1 + 0,05) = 0,0476 \text{ of plastic}$$

$$1,00 / (1 + 0,05) = 0,9524 \text{ of bread}$$

A tone of this waste flow then means 47,6kg of plastic + 952,4kg of bread.

All composing materials need to be present in the Material list. The material list can be adapted based on the requirements of the data at any time during the project. Only those materials that have already been assigned to some of the flows cannot be altered anymore. New materials can be added at any time at any of the hierarchical levels.

Fractions for one composition always need to add up to 1.00 (like in the example in the template 0.7 of Wood + 0.3 of Steel = 1.0 of 02212 Roofing materials)

The column “NACE” refers to the primary activity as producer of this particular composition.

Composition names should always be unique, however the same composition can be used in multiple flows and stocks. E.g. if a general composition for municipal solid waste is known for the whole study area and no further differentiation is possible between different municipalities/city districts, then the composition can be called “General municipal solid waste” and referred to for each of the actors of the household type following there amounts.

If known, the name of the composition should start with an EWC code.

Each material should be tagged with Avoidable (TRUE), e.g. a banana, or Unavoidable (FALSE), e.g. a banana peel. If this tag is not applicable, then the material is unavoidable and therefore (FALSE).

Additional descriptions can be given for each individual flow defining collection method, quality, additional materials, processing, etc. in the table of Flows and Stocks.

See section “Data Sources” for the explanation of the “Source” column.

Flows & Stocks

Unique Key: [Origin, Destination, Composition]

Origin	Destination *	Amount	Composition	Year	Waste	Source	Description	Process*
<i>BvDid or other identifier (must be present in the list of Actors)</i>	<i>BvDid or other identifier (must be present in the list of Actors)</i> <i>*No destination for stocks</i>	<i>t/year</i>	<i>Name of the composition, must be present in the list of Compositions</i>		<i>TRUE if it is waste, FALSE if it is product</i>	<i>BibTEX key for the source (must be present in the list of publications)</i>		<i>Treatment process, should be present in the list of processes provided by the WP4 (optional)</i>
LMA00001	LMA00024	900	02212 Roofing materials	2016	TRUE	Ima2016	Contains asbestos	Recycling

Flows are the most important part of the AS-MFA framework as they demonstrate how the materials are flowing from one actor to another.

Stocks demonstrate where materials accumulate, i.e. where they stay for longer than a year. The only difference between the Flow and Stock tables is that Stocks do not have destinations, which means that materials are kept at the location of the actor of origin.

Flows are always moving in a direction from the origin to the destination. The amounts are added in **tonnes per year** (t/year).

Both the origin and the destination are actors that must be present in the Actor table. If flows need to involve actors that are not yet present in the actor table, those new actors need to be added first. The origin and the destination columns refer to the unique identifier of an actor that in most cases is the BvDid number taken from the ORBIS database, otherwise it can be an identifier of the administrative area (in the case of households) or a unique identifier given by the data captain (see more detailed explanations in the section “Actors”).

Composition refers to the unique name of a composition that has been provided in the table “Composition”. Multiple flows and stocks can refer to the same composition.

It is possible that flows between the same origin and the same destination have different waste compositions (e.g. a bakery throws away 10t of bread packed in plastic and 20t of bread packed in paper). In that case this information can be added as two separate flows referring to two different compositions, or as one flow with the composition of [bread; plastic; paper] as long as the multiplication of ratios and amounts lead to a correct result for each of the separate material fractions.

Each flow must have a unique combination of Origin-Destination-Composition. If there are two or more flows with the same combination, they must be added up together before entering the system, otherwise one may overwrite the other. Each stock must have a unique combination of Origin-Composition.

Each flow/stock should be tagged as waste or not waste according to the definition developed in REPAiR.

Each flow/stock can be described in a free text format providing any information that may be relevant later, e.g. quality of the product/waste, treatment process, transportation or collection process or any other information that is present in the data however not supported by the GDSE data structure.

The “process” column refers to the treatment process of waste, e.g. recycling, composting, biodegrading, etc. The definite list of processes will be provided by the WP4, until now the process name can be entered in a free form.

See section “Data Sources” for the explanation of the “Source” column.

Flows and stocks should be written in separate .tsv files with appropriate naming.

Data Sources

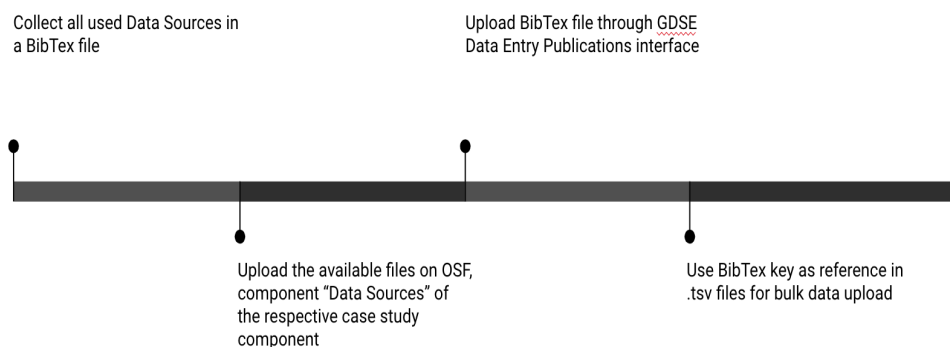


Figure 37: Data Sources

To ensure scientific integrity and in order to prevent mistakes and facilitate their eventual correction, every number that is used for the AS-MFA needs to be traceable back to its original source. Therefore, the data structure supports referencing of each flow and composition.

Flows and compositions may have separate data sources, e.g. the amounts of household waste generated in a certain municipality can be provided by the municipality, while the composition of household waste may be researched by a different institution and for a different purpose.

The process to control and keep track of the used data sources is as explained in the timeline above.

All used data sources must be described using a BibTex format (more information available here: <http://www.bibtex.org/>). We recommend using Mendeley, JabRef or a similar bibliography reference management software to compile a list of used sources.

The BibTex file needs to be uploaded into the database **before** uploading the Flows and Compositions following these steps:

1. Login into the GDSE on <https://gdse.h2020repair.bk.tudelft.nl/>
2. Use this link to enter the publication:
https://gdse.h2020repair.bk.tudelft.nl/admin/publications_bootstrap/publication/
3. Click on “IMPORT BIBTEX +” button on the top right of the screen
4. Copy and paste your BibTex(es) into the empty field - you can paste more than one just as it would appear in a .bib file
5. Click “IMPORT”
6. If the import has been successful, your publications will appear on the list. Use the **citekey** as a reference in the Composition and Flow tables.

Alternatively, each source can be entered manually by clicking “ADD PUBLICATION +” button and filling all relevant fields manually. Citekey will generated automatically unless provided by the user.

If a source does not have a permanent identification (DOI or POI), it needs to be uploaded on the OSF in the folder “Data Sources”. The folder is only accessible to the administrators of OSF (Max Bohnet, Rusne Sileryte and Alex Wandl) and the respective data captains of the case study, therefore it is completely safe to upload even sensitive and protected datasets there.

Most Common Errors

- An Activity refers to the activity group that is not in the Activity Group list
- An Actor refers to the activity that is not in the Activity list
- A Flow/Stock refers to Actor or Material that are not in their respective lists
- Material fractions do not add up to 1.00 in Product/Waste composition
- Household administrative unit missing in the list of Administrative Units previously uploaded by the data captains (check on OSF for the files that have been used)
- “Tabs” in texts
- Duplicate entries
- Shapefiles not in WGS84
- Amounts not in tonnes/year but kg/year
- Missing data sources
- Software conversion/interpretation problems, e.g. SHP to XLSX that turn postcodes into numbers (removing the “0” in front), interpret thousand separators as decimals or the other way round
- Origin and destination is the same within one flow

Correspondence between the templates and the GDSE database structure (for WP2)*Table 19: Dataset Administrative Units*

Dataset	Administrative Units																												
Data File	OSF/--CaseStudy--/Spatial_data/T3.1_Administrative_Units/--UnitName--.shp																												
Tables	studyarea_adminlevels studyarea_area																												
Correspondence	<p>'Name' → studyarea_adminlevels.name 'Administrative level' → 'studyarea_adminlevels.level' 'Level ID' → studyarea_adminlevels.casestudy_id* File Name' → shapefile for the studyarea_area</p> <p>\$geom(multipolygon) → studyarea_area.geom (W GS84, 4326) 'Name' → studyarea_area.name 'AdminLevel' → studyarea_area.adminlevel_id 'Code' → studyarea_area.code**</p> <p>*</p> <table> <tr> <td>ID</td><td>Level</td></tr> <tr> <td>1</td><td>World</td></tr> <tr> <td>2</td><td>Continent</td></tr> <tr> <td>3</td><td>Country</td></tr> <tr> <td>4</td><td>NUTS1</td></tr> <tr> <td>5</td><td>NUTS2</td></tr> <tr> <td>6</td><td>NUTS3</td></tr> <tr> <td>7</td><td>District</td></tr> <tr> <td>8</td><td>Municipality</td></tr> <tr> <td>9</td><td>CityDistrict</td></tr> <tr> <td>10</td><td>CityNeighbourhood</td></tr> <tr> <td>11</td><td>CityBlock</td></tr> <tr> <td>12</td><td>StreetSection</td></tr> <tr> <td>13</td><td>House</td></tr> </table> <p>** e.g. NUTS code or any other administrative code corresponding to the area</p>	ID	Level	1	World	2	Continent	3	Country	4	NUTS1	5	NUTS2	6	NUTS3	7	District	8	Municipality	9	CityDistrict	10	CityNeighbourhood	11	CityBlock	12	StreetSection	13	House
ID	Level																												
1	World																												
2	Continent																												
3	Country																												
4	NUTS1																												
5	NUTS2																												
6	NUTS3																												
7	District																												
8	Municipality																												
9	CityDistrict																												
10	CityNeighbourhood																												
11	CityBlock																												
12	StreetSection																												
13	House																												
Template	https://osf.io/9x27b/ → Administrative Levels																												

Table 20: Dataset Materials

Dataset	Materials
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Materials_--Keyflow--/T3.2_Materials.tsv

Tables	asmfa_material
Correspondence	\$column# → asmfa_material.level \$text → asmfa_material.name \$text(column#-1) → asmfa_material.parent
Template	https://osf.io/68m5e/

Table 21: Dataset Activity Groups

Dataset	Activity Groups
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Activity_groups_--Keyflow--/T3.2_Activity_Groups.tsv
Tables	asmfa_activitygroup
Correspondence	'Code' → asmfa_activitygroup.code* 'Name' → asmfa_activitygroup.name *ISIC Rev.4/NACE Rev. 2, letters of the NACE code
Template	https://osf.io/wrpzx/

Table 22: Dataset Activity

Dataset	Activities
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Activities_--Keyflow--/T3.2_Activities.tsv
Tables	asmfa_activity
Correspondence	'NACE' → asmfa_activity.nace* 'Name' → asmfa_activity.name 'AG' → asmfa_activity.activitygroup_id *NACE Rev. 2, in form of 'X-0001'
Template	https://osf.io/pfu3e/

Table 23: Dataset Actors

Dataset	Actors ¹
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Actors_--Keyflow--/T3.2_Actors.tsv
Tables	asmfa_actor
Correspondence	'BvD ID number' → asmfa_activity.BvDid" 'Company name' → asmfa_activity.name 'Cons.code' → asmfa_activity.consCode 'Lastavail.year' → asmfa_activity.year 'Trade description (English)' → asmfa_activity.description_eng 'Trade description in original language' → asmfa_activity.description 'BvD Indep. Indic.' → asmfa_activity.BvDii 'Website address' → asmfa_activity.website 'Number of employeesLast avail. yr' → asmfa_activity.employees EUR' → asmfa_activity.turnover_currency 'Operatingrevenue(Turnover)th EURLast avail. yr' → asmfa_activity.turnover 'NACE Rev. 2Core code (4 digits)' → asmfa_activity.activity_id
Template	No template available, ORBIS output table with all required fields

¹ Uploaded by WP2 as an extract from ORBIS database

Table 24: Dataset Geolocated Actors

Dataset	Geolocated Actors ²
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Actors_geolocated_--Keyflow--/T3.2_Actors_geolocated.shp
Tables	asmfa_administrativelocation
Correspondence	'Postcode' → asmfa_administrativelocation.postcode 'Address' → asmfa_administrativelocation.address 'City' → asmfa_administrativelocation.city 'BvDIDNR' → asmfa_administrativelocation.actor_id Point(x,y): → asmfa_administrativelocation.geom (WGS84, 4326)
Template	No template available

Table 25: Dataset Households

Dataset	Households
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Actors_--Keyflow--/T3.2_Households.tsv
Tables	asmfa_actor
Correspondence	'Identifier*': → asmfa_actor.BvDid 'Name': → asmfa_actor.name V-0000 → asmfa_actor.activity_id *Identifier must match an identifier among the provided Administrative Units, metadata file should indicate which level of Administrative Units does the identifier refer to
Template	https://osf.io/nc36f/

Table 26: Dataset Filtered actors

Dataset	Filtered actors
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Actors_--Keyflow--/T3.2_Actors_filtered.tsv
Tables	asmfa_actor

² Uploaded by WP2 as a result of Geocoding service

Correspondence	'Identifier': → asmfa_actor.BvDId 'Reason*': → asmfa_actor.reason_id 'Included' → asmfa_actor.included *0 - Included; 4 - Outside material Scope; 5 - Does not produce waste
Template	https://osf.io/mjh5g

Table 27: Dataset Waste/ Product Composition

Dataset	Waste/Product Composition
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Compositions_--Keyflow--/T3.2_Waste_composition.tsv OSF/--CaseStudy--/MFA_data/T3.2_Compositions_--Keyflow--/T3.2_Product_composition.tsv
Tables	asmfa_composition asmfa_productfraction
Correspondence	'Custom name' → asmfa_composition.name 'NACE' → asmfa_composition.nace 'Fraction' → asmfa_productfraction.fraction* 'Material' → asmfa_productfraction.material_id 'Avoidable' → asmfa_productfraction.avoidable *all product fractions referring to the same composition must add up to 1,00
Template	https://osf.io/h8t5f/

Table 28: Dataset Flows

Dataset	Flows
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Flows_--Keyflow--/T3.2_Flows_actor2actor.tsv
Tables	asmfa_actor2actor
Correspondence	'Amount' → asmfa_actor2actor.amount 'Destination' → asmfa_actor2actor.destination_id 'Origin' → asmfa_actor2actor.origin_id 'Year' → asmfa_actor2actor.year 'Source' → asmfa_actor2actor.publication_id* True/False → asmfa_actor2actor.waste 'Composition' → asmfa_actor2actor.composition_id *Source is provided as a bibtex key, respective bibtex entry must be uploaded to the database beforehand
Template	https://osf.io/2t67r/

Table 29: Dataset Stocks

Dataset	Stocks
Data File	OSF/--CaseStudy--/MFA_data/T3.2_Stocks_--Keyflow--/T3.2_Stocks_actor.tsv
Tables	asmfa_actorstock
Correspondence	'Amount' → asmfa_actor2actor.amount 'Origin' → asmfa_actor2actor.origin_id 'Year' → asmfa_actor2actor.year 'Source' → asmfa_actor2actor.publication_id* True/False → asmfa_actor2actor.waste 'Composition' → asmfa_actor2actor.composition_id *Source is provided as a bibtex key, respective bibtex entry must be uploaded to the database beforehand
Template	https://osf.io/vkfb3

Common

Datasets common for all case studies. Prepared and uploaded by WP2.

Table 30: Dataset NACE-EWC correspondence table

Dataset	NACE-EWC correspondence table
Data File	OSF/Common/T3.2_AS-MFA/NACE-EWC.tsv
Tables	asmfa_waste, asmfa_composition
Correspondence	'NACE' → asmfa_composition.nace 'Haz' → asmfa_waste.hazardous 'EWC_code' → asmfa_waste.ewc 'EWC_name' → asmfa_composition.name 'Item_descr' → asmfa_waste.wastetype

Table 31: NACE-CPA correspondence table

Dataset	NACE-CPA correspondence table
Data File	OSF/Common/T3.2_AS-MFA/NACE-CPA2008.tsv
Tables	asmfa_product, asmfa_composition
Correspondence	'NACE' → asmfa_composition.nace 'CPA 2008 DESCRIPTION' → asmfa_composition.name 'CPA 2008 CODE' → asmfa_product.cpa

References

Arciniegas, G., Šileryté, R., Dąbrowski, M., Wandl, A., Dukai, B., Bohnet, M., & Gutsche, J.-M. (2019). A geodesign decision support environment for integrating management of resource flows in spatial planning. *Urban Planning*, 4(3), 32–51.

<http://dx.doi.org/10.17645/up.v4i3.2173>

Jochim, V. (2018): Entwicklung eines Visualisierungskonzeptes zur Darstellung georeferenzierter Mengenflüsse im Rahmen des EU-Projektes REPAiR. Master Thesis at the Department of Geoinformatics at HafenCity University.

REPAiR (2016). Deliverable 2.1: Vision of the GDSE Applications. Project REPAiR.

Retrieved from: [http://h2020repair.eu/wp-](http://h2020repair.eu/wp-content/uploads/2017/09/Deliverable_2.1_Vision_of_the_GDSE_Applications.pdf)

[content/uploads/2017/09/Deliverable_2.1_Vision_of_the_GDSE_Applications.pdf](http://h2020repair.eu/wp-content/uploads/2017/09/Deliverable_2.1_Vision_of_the_GDSE_Applications.pdf)

REPAiR (2017a). Deliverable 2.2: Data Requirement Description and Data Delivery Plan for the Case Study Areas. Project REPAiR.

REPAiR (2017b). Deliverable 2.3: Programmed GDSE Modules. Project REPAiR.

REPAiR (2019a). Deliverable 2.4: Handbook for Geodesign Workshops. Project REPAiR.

Retrieved from:

https://teams.connect.tudelft.nl/projects/vc/repair/Deliverables/D2.4%20Handbook%20for%20Geodesign%20Workshops_final_version.pdf

REPAiR (2019b). D3.3 Process model for the two pilot cases: Amsterdam, the Netherlands & Naples, Italy. Project REPAiR.

REPAiR (2020). Deliverable 2.5: Adapted GDSE modules. Project REPAiR.