The recipe names an application, but the Cookbook uses that name only to look up which capability the application exemplifies, and then discards it. What the user is offered is every application registered in **their** hub that declares the same capability. In Figure 16 the Rotterdam hub offers Rana Water Intelligence at €500 per execution alongside AquaGuard Flood Modeller at €480, both marked API verified. Neither is privileged by the recipe.

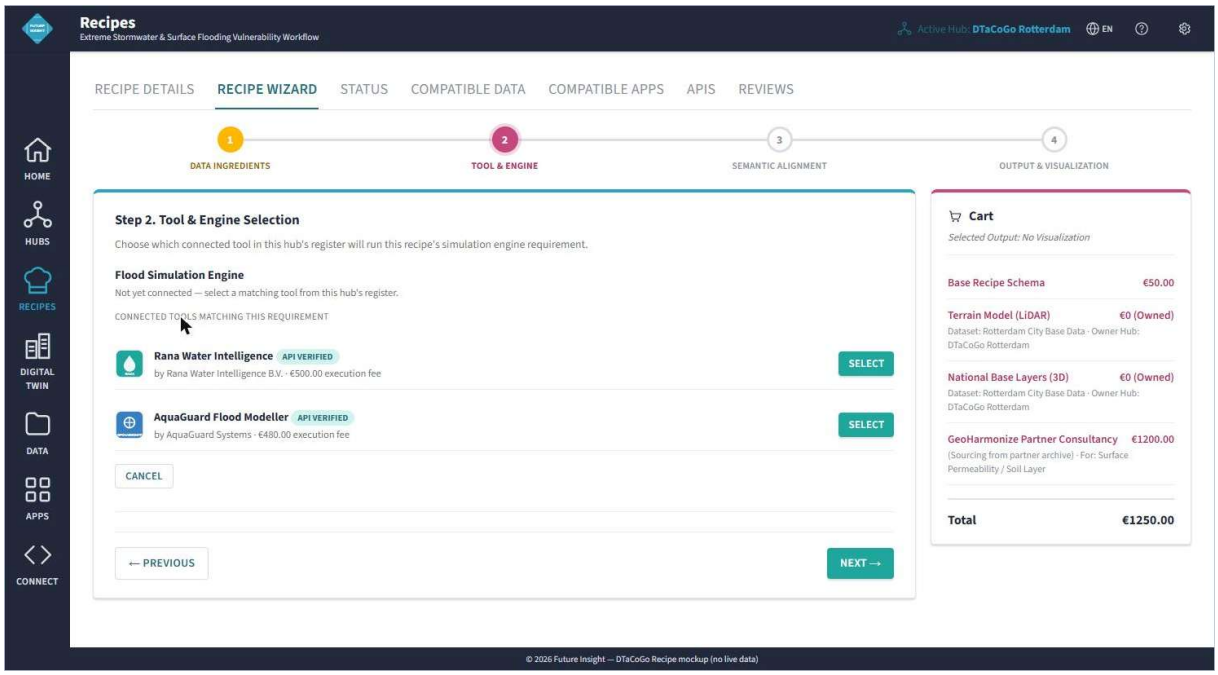


Figure 16 — Step 2. Both registered engines offering the required capability are candidates; the recipe's named application receives no preference. Route: #/recipes/recipe-rana/wizard

This is why a recipe survives the journey between cities. 's-Hertogenbosch or Tampere need not license Rana in order to use a recipe Rotterdam authored with Rana; they need a registered engine that declares the flood simulation capability and satisfies the recipe's execution pattern. The recipe itself is not edited.

4.4 Step 3 — Semantic alignment

Having a dataset in the right format does not mean the tool can read it. The format governs the container; the attributes inside it are a separate matter, and this is exactly the problem the Common Information Model and the Purple Polar Bear layer exist to solve, as set out in D3.1.

Step 3 takes each dataset-and-tool pairing and asks whether a mapping exists. Where PPB already holds one, it is offered by name and applied in a single action. In Figure 18 the pairing of building footprints with Rana resolves to the predefined mapping *3D Tiles (OGC) → Rana Schematisation Engine*, attributed to PPB. Where no predefined mapping exists, the user configures one by hand.

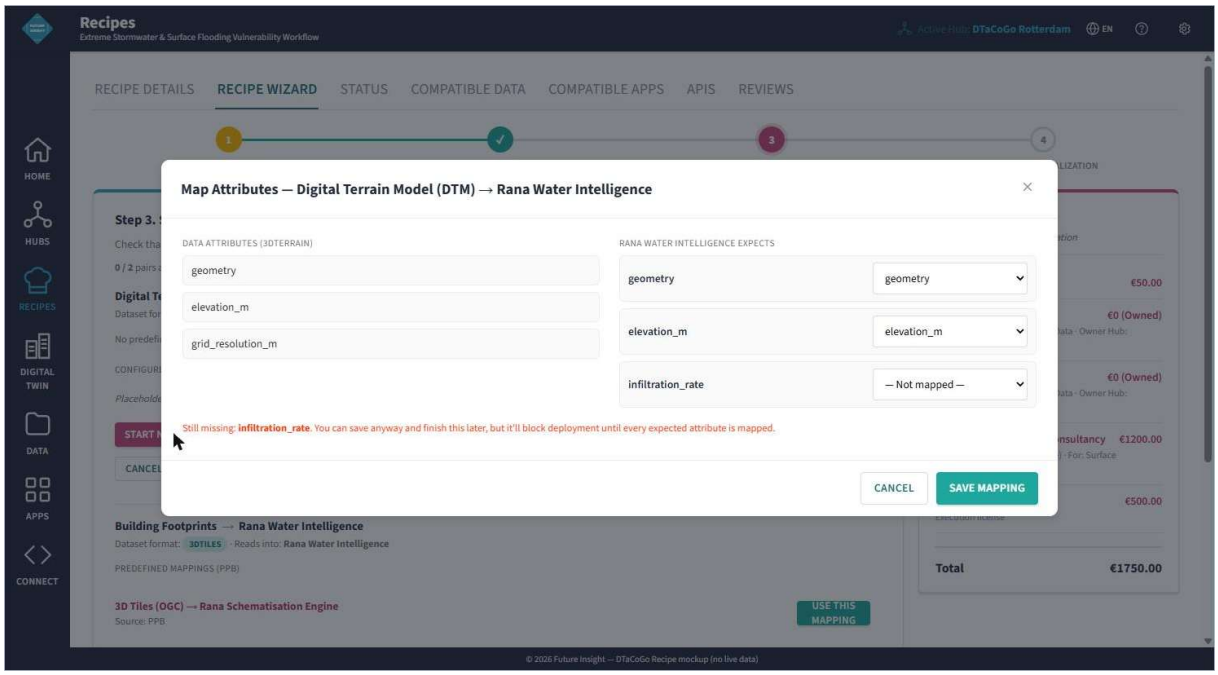


Figure 17 — Attribute mapping. The dataset's own attributes on the left, the tool's expected fields on the right, and an explicit report of what is still unmapped. Route: #/recipes/recipe-rana/wizard

The mapping dialogue in Figure 17 is worth reading closely, because it embodies the plan-first principle. Two of the three fields Rana expects have been mapped; *infiltration_rate* has not, and the dialogue says so by name. It also states the consequence — the mapping may be saved and finished later, but deployment will be blocked until every expected attribute is mapped. A half-finished mapping is a legitimate intermediate state, and it is tracked rather than discarded.

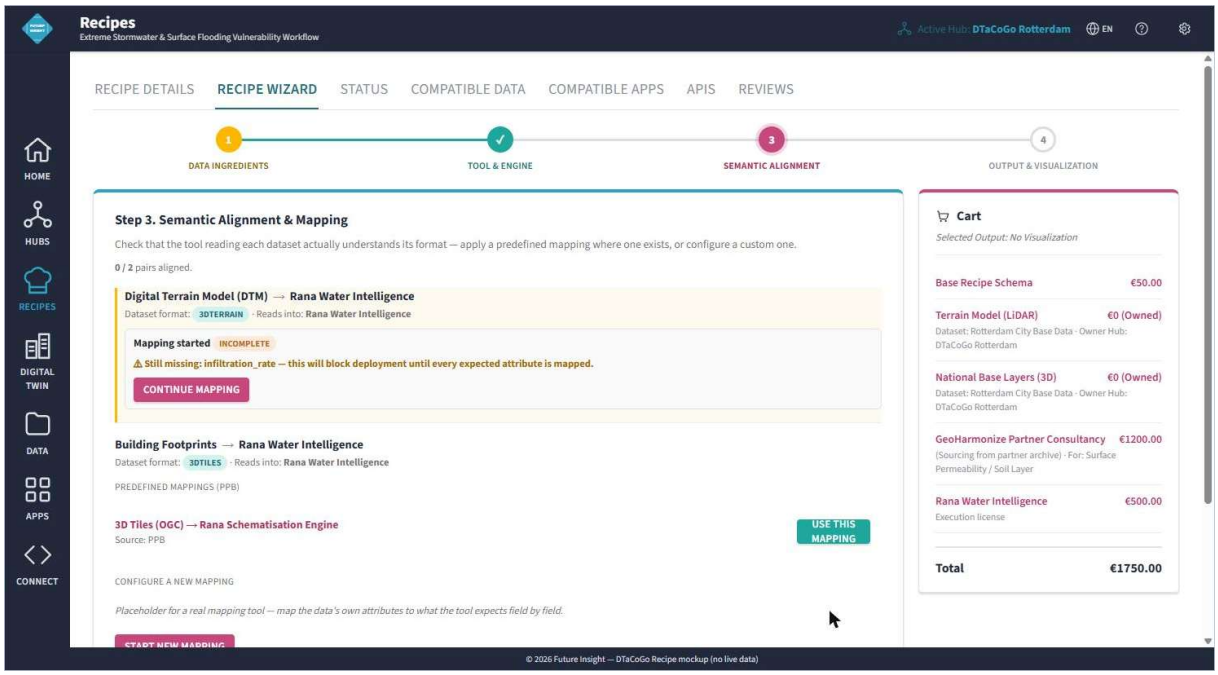


Figure 18 — Step 3 with one pairing incomplete and one resolved through a PPB predefined mapping. Route: #/recipes/recipe-rana/wizard

There is a substantive point buried in this example. *infiltration_rate* is not an attribute of a terrain model at all; it belongs to the surface permeability layer. A production Cookbook should be able to satisfy a tool's expected field from any bound dataset rather than only from the one currently being mapped. This is recorded in chapter 9. Status at project closure: open. The problem is wider than the example suggests, because matching concerns not only which attribute satisfies an expected field but also whether that attribute's values are compatible. The behaviour shown in the mockup is for demonstration only, and the question needs further investigation. For now, the workable solution is the PPB service layer, which provides and maintains these mappings outside the Cookbook (D3.1).

4.5 Step 4 — Output and visualisation

The last step asks two questions. Where should the result be written — which hub, which dataset — with the output format fixed by the recipe and displayed as read-only. And should a visualisation experience be added on top, or is a data export sufficient?

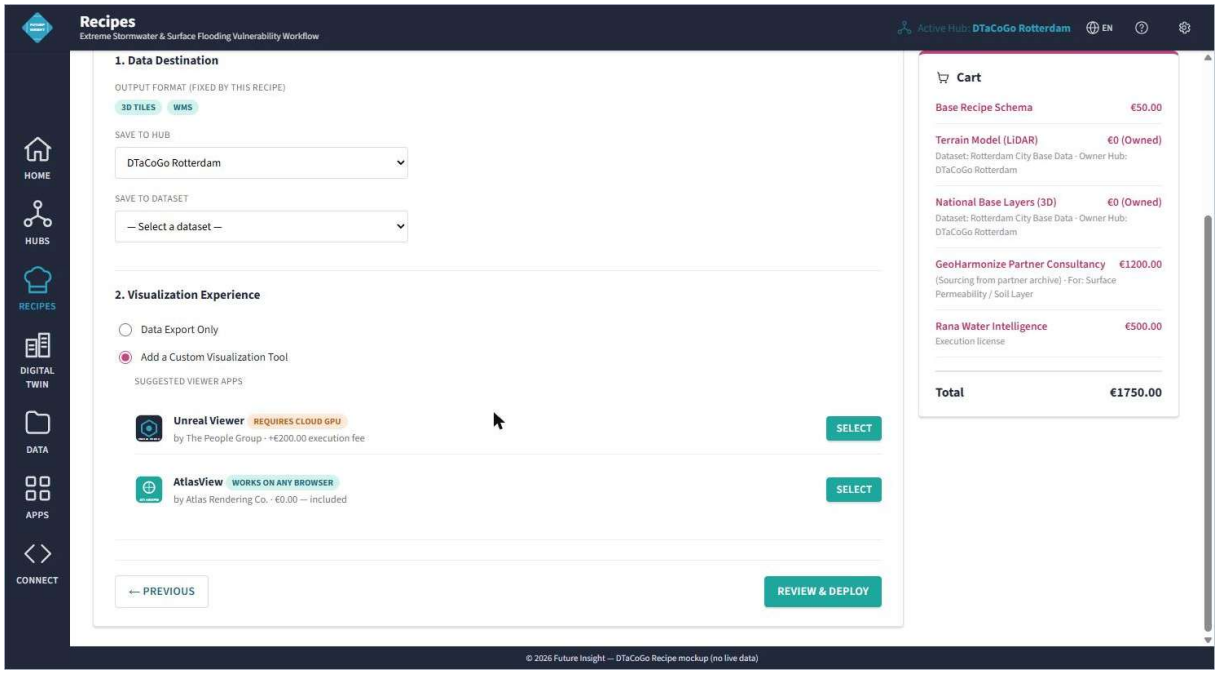


Figure 19 — Step 4. The output format is fixed by the recipe; the viewer is the user's choice. Route: #/recipes/recipe-rana/wizard

The viewer choice is the same capability indirection applied to visualisation. The Unreal Viewer supplied by The People Group requires a cloud GPU and carries a €200 execution fee; AtlasView works in any browser and is included. A city without GPU budget is not thereby excluded from the recipe — it gets a less immersive result. This is also the point at which the plug-in discussed in chapter 5 enters the configuration.

4.6 The two-severity readiness model

Readiness is the wizard's central mechanism, and it distinguishes two severities rather than treating everything unfinished as equally unfinished.

Severity	Condition	Effect
Blocked	No decision has been taken for this requirement at all.	Stops the user advancing to the next step, and blocks deployment.
Pending	A decision has been taken and a plan exists — a	Permits the user to continue

Severity	Condition	Effect
	partner engaged, or sourcing declared to be handled outside the platform — but no correctly formatted resource has yet been registered.	planning and costing. Blocks deployment only.
Ready	A correctly formatted resource is registered and, where relevant, semantically mapped.	Blocks nothing.

Table 3 — The two severities of unreadiness, and what each prevents.

The distinction earns its keep because it matches how cities actually work. A department can complete a configuration, obtain a total cost, and take that figure to a budget holder while three of its datasets are still being procured. Collapsing the two severities would force a choice between a system that blocks all planning until every resource exists, and one that lets an unrunnable configuration be declared ready.

Warnings are withheld until the user tries to move on. Figure 20 shows the effect of pressing Next with two slots undecided: the offending slots are outlined and given a specific instruction. A user who has not yet reached a slot is not reprimanded for not having reached it.

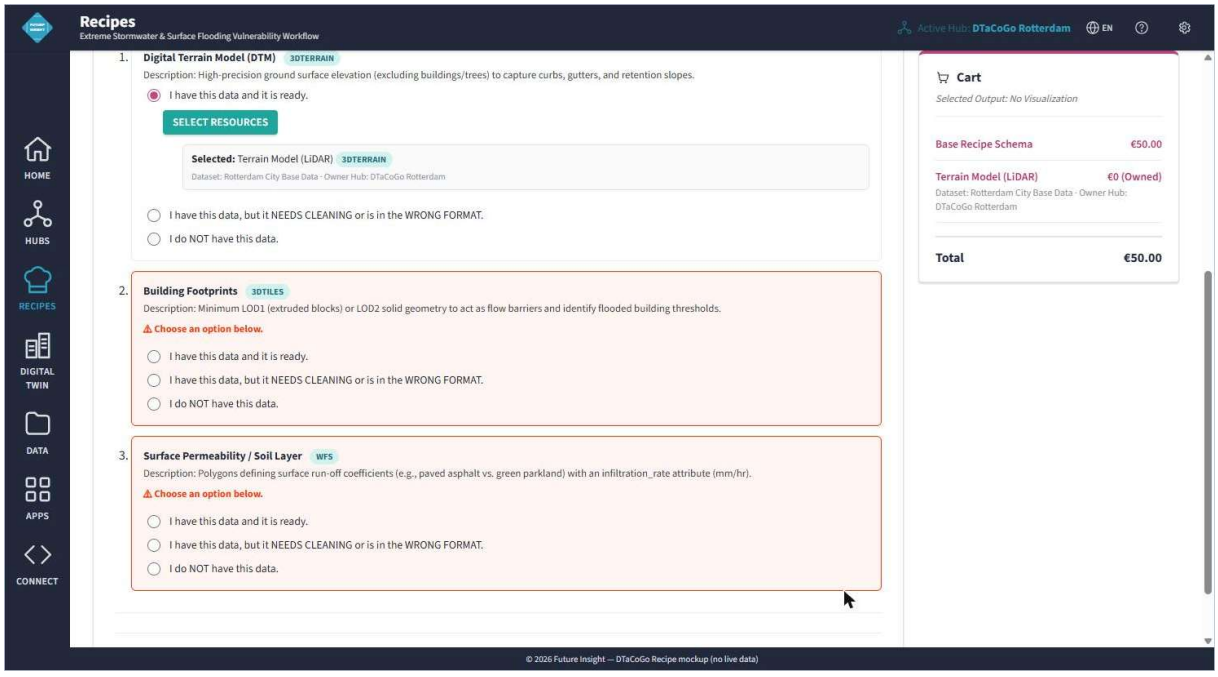


Figure 20 — Blocked requirements revealed on attempting to advance, each reported against its own slot. Route: #/recipes/recipe-rana/wizard

A pending requirement looks different, and deliberately so: amber rather than red, with the plan recorded, the outstanding action named, and the consequence stated.

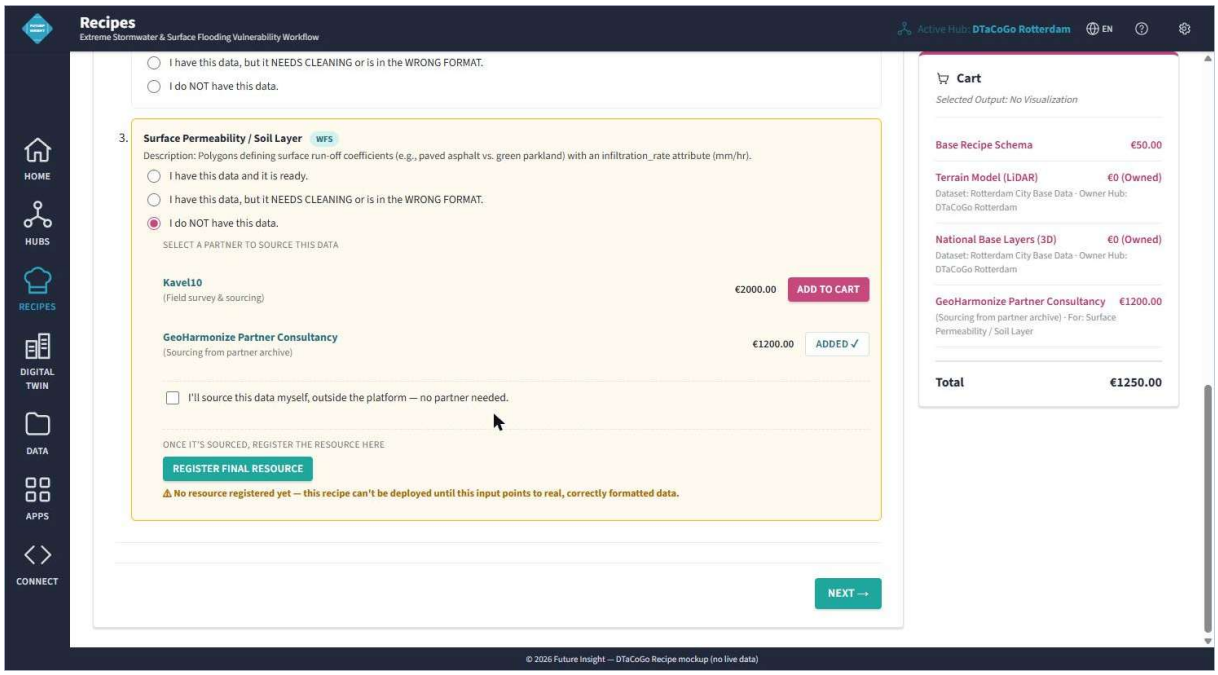


Figure 21 — A pending requirement: a partner is engaged and priced, and the outstanding registration is stated explicitly. Route: #/recipes/recipe-rana/wizard

The Status tab aggregates all of this into one place. It carries a single deployment badge — here reading **Cannot be deployed yet** — and then separates **Blocking the wizard** from **Blocking deployment**, naming the specific slot, pairing or attribute responsible for each entry. This is the view a project lead would take to a steering meeting.

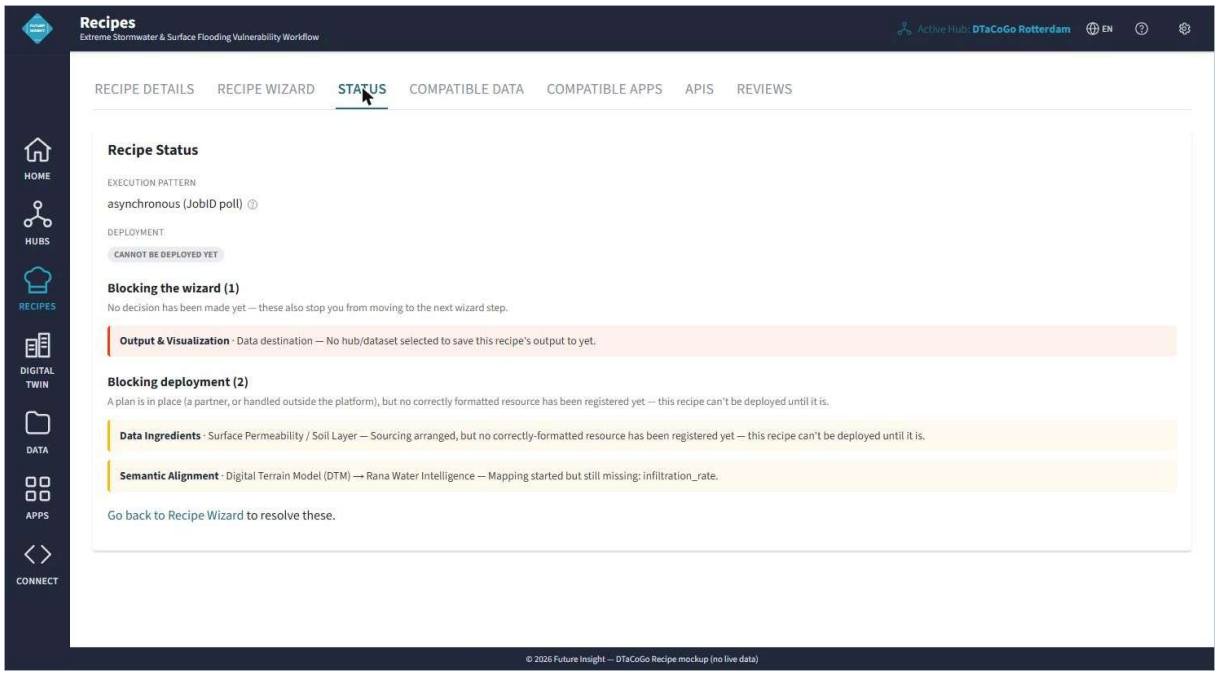


Figure 22 — The Status tab: one blocking item, two deployment-blocking items, each named precisely. Route: #/recipes/recipe-rana/status

4.7 The cost model

A running cart accumulates every cost implication of the decisions taken, and is visible throughout. It carries a base recipe schema fee; owned resources at zero, annotated with their dataset and owning hub so that the zero is auditable; partner services at their quoted price; the execution licence of the chosen engine; and the visualisation fee where a paid viewer is selected.

Cart line	Origin	Example from the walkthrough
Base recipe schema	Fixed fee for instantiating the recipe	€50.00
Bound resource	Zero where the consuming hub already owns it	Terrain Model (LiDAR) — €0 (Owned)
Partner service	Quoted price of a sourcing or transformation partner	GeoHarmonize Partner Consultancy — €1,200.00
Execution licence	Per-execution fee of the bound engine	Rana Water Intelligence — €500.00
Visualisation	Fee of the selected viewer application	Unreal Viewer (The People Group) — €200.00

Table 4 — The cost lines accumulated during the walkthrough, totalling €1,950.00 for the configuration shown.

Showing owned resources at zero rather than omitting them is a deliberate choice: it demonstrates that the requirement was met and by what, which is what makes the total defensible to a budget holder. All figures in the mockup are illustrative.

5 Where scenario parameters are set

5.1 A boundary, not a gap

Nothing in the Cookbook lets a planner type a rain intensity. There is no screen on which one sets a 70 mm/hr design storm and presses go. Read casually, that looks like the central promise of the recipe concept going unfulfilled. It is not. It is a division of responsibility between two layers, and it is the right one.

Scenario variables are set in the visualisation layer. The user opens the Unreal Digital Twin application, loads the twin, starts the plug-in for the relevant model, picks the calculation method and defines its variables there. The User tab runbook in Figure 10 states this as step 6, and the Development tab names the SDK-light through which a provider implements such a plug-in as obligation 6.

The reasoning is sound. Choosing a rain intensity is not a configuration decision taken once when an ecosystem is stood up; it is an analytical decision taken repeatedly, in context, while looking at the result. A planner comparing a 40 mm/hr event with a 70 mm/hr event wants to see the water rise on the model in front of them, not to re-enter a catalogue, re-open a wizard and re-deploy a configuration. Putting the control where the result is displayed is what makes iteration possible.

The consequence for this deliverable is a clean statement of scope. The Cookbook publishes the parameter **contract**. The viewer plug-in provides the parameter **controls**. The recipe is the interface between them.

Concern	Owned by	Rationale
Which variables exist, their type, unit and default	The Cookbook, via the recipe record	Part of the recipe's portable definition; must travel between cities unchanged.
The value of a variable for a particular run	The Digital Twin viewer plug-in	An analytical decision taken repeatedly while examining the result.
Rendering an appropriate control for each variable	The Digital Twin viewer plug-in, from the published contract	The viewer knows its own interaction idiom; the recipe should not dictate it.
Recording which values produced which result	Run provenance, returning to the catalogue	Needed for reproducibility; out of scope for the current mockup.

Table 5 — Division of responsibility for scenario parameters.

5.2 What the recipe publishes today

The stormwater recipe's Orchestrator projection declares four parameters: a dataset reference for the schematisation, a simulation duration in hours defaulting to 24, a rain duration in minutes defaulting to 60, and a rain intensity in mm/hr defaulting to 40. Two of these are typed as sliders, which is an explicit instruction to the viewer about the kind of control to present.

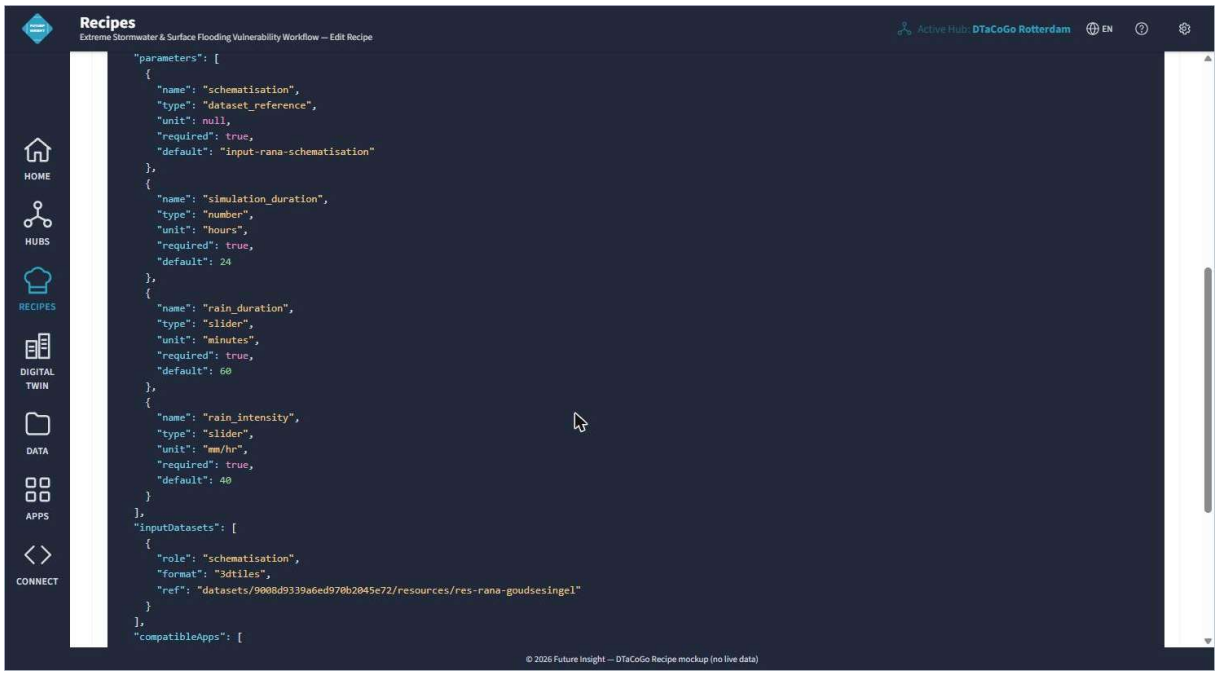


Figure 23 — The parameter block of the machine-readable record. Route: #/recipes/recipe-rana/edit/orchestrator

The wizard also surfaces these constraints in prose at step 1, under execution constraints: a standardised event length, and a design storm precipitation input. A user configuring the ecosystem therefore knows which variables they will later be asked for, without being asked for them prematurely.

5.3 A defect in the contract

The published projection emits *name*, *type*, *unit*, *required* and *default* for each parameter. It does not emit the bounds. For a parameter typed as a slider this is a broken contract: a plug-in that is told to render *rain_intensity* as a slider in mm/hr with a default of 40 has no way to know whether the track should run to 100 or to 250, or what step size is meaningful.

The practical effect is that every viewer implementation must hard-code ranges that properly belong to the recipe — which reintroduces exactly the per-city, per-tool coupling the recipe was designed to remove, and does so silently. The fix is small and should be made before any plug-in is written against this contract:

- Carry *min*, *max* and *step* on every numeric and slider parameter, and carry the permitted values on every enumerated one.
- Treat the bounds as required for *type: slider*, and validate this when a recipe is published.
- State, for each parameter, whether it is fixed by the recipe or free for the user to set at run time — the current record does not distinguish these.

This is recorded as a blocking item for the plug-in workstream in Table 10. Status at project closure: unresolved. Carrying bounds is work in progress in the context of OGC API — Processes, and must then be implemented in the Unreal Digital Twin application.

6 Publishing recipes

6.1 The recipe record

A published recipe carries four groups of fields.

Group	Fields	Purpose
Identity and conformance	*id*, *type*, *conformsTo*, *title*, *description*	Stable reference and declared conformance to the D4.1 classes.
Governance	*owner*, *findability*, *license*	Who owns the record, who may see it, on what terms it may be used.
Discovery	*projectPhase*, *targetRoles*, *theme*, *location*, *executionPattern*	The facets on which the catalogue searches; advisory, not authorising.
Execution contract	*parameters*, *inputDatasets*, *compatibleApps*, output formats	What must be bound, and what the result will be.

Table 6 — The four field groups of a recipe record.

6.2 The publication lifecycle

A recipe is a governed publication, not a file that is circulated by e-mail. It moves through five states, and each transition has an identified authoriser.

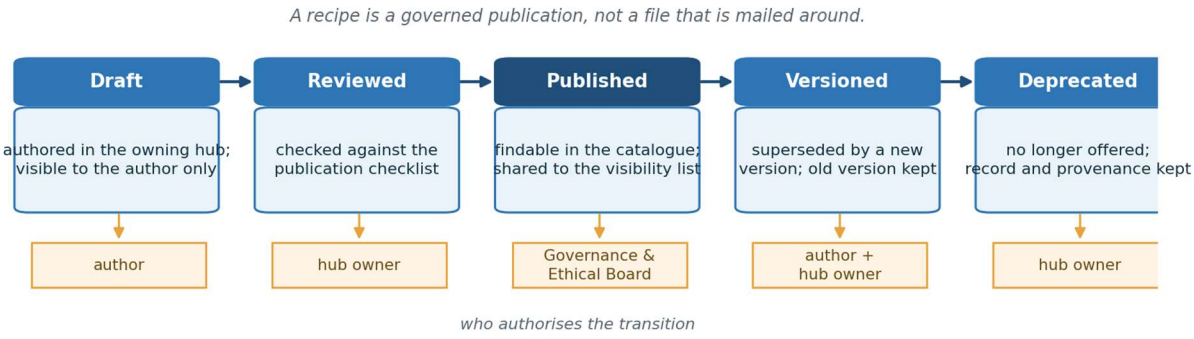


Figure 24 — The publication lifecycle of a recipe and its authorising parties.

The publication gate is where the governance and ethics process of D2.3 attaches. A recipe that binds a model carrying ethical constraints identified in D3.6 cannot be published without those constraints being carried forward onto the recipe record. Deprecation removes a recipe from circulation but does not delete it: configurations derived from it remain traceable to the version they were derived from.

6.3 Metadata mapping

For the Cookbook to federate beyond DT-aCoGo, its records must be expressible in standards that other catalogues already speak. The intended targets are OGC API — Records for the catalogue interface and DCAT-AP-NL for national alignment.

Recipe field	OGC API — Records	DCAT-AP-NL

Recipe field	OGC API — Records	DCAT-AP-NL
id	*id*	*dct:identifier*
title	*properties.title*	*dct:title*
description	*properties.description*	*dct:description*
owner	*properties.providers*	*dct:publisher*
license	*properties.license*	*dct:license*
theme	*properties.themes*	*dcat:theme*
location	*geometry / properties.extent*	*dct:spatial*
conformsTo	*conformsTo*	*dct:conformsTo*
inputDatasets	*properties.associations*	*dct:relation*

Table 7 — Indicative mapping of the recipe record onto OGC API — Records and DCAT-AP-NL. Not yet implemented; see chapter 9.

6.4 Versioning, and the relation to the nLDT Cookbook

A recipe changes when its requirements change — a new input becomes necessary, a format is superseded, a parameter's meaning shifts. Such a change must produce a new version rather than mutate the existing record, because configurations elsewhere were derived from what the record said at the time. Superseded versions remain resolvable.

The Dutch national digital twin programme uses the same three-part vocabulary: a Cookbook holding recipes, a Cook executing them, and an App Store supplying the components they reference. DT-aCoGo's contribution is not a competing definition but a worked instance: a Cookbook populated with real sectoral recipes, applied across three cities in two countries, resolved against a live semantic layer. The conformance classes in the Orchestrator projection are the seam along which the two are intended to join.

7 Governance, ethics and trust

7.1 Ownership and visibility

Every recipe has an owning hub, displayed on the card and on the record. Ownership determines who may edit and who may publish. Findability — public, shared with a named list of hubs, or private — determines who may see the record. These are properties of the recipe record, and they are separate from access to the resources the recipe references: a public recipe may reference datasets that a given reader cannot open.

That separation is deliberate and should not be softened. It is what allows a recipe to function as a publishable method while the data it consumes remains governed by the owning organisation.

7.2 Roles are advisory

The *targetRoles* field uses the vocabulary defined in D2.1 — in the stormwater recipe, project engineer and project coordinator. This is guidance about who will find the recipe useful. It is **not** an authorisation statement, and the Cookbook must never be implemented as though it were. Access control is defined in D2.3 and enforced by the hub. A reader whose role is not listed is not prevented from using the recipe; a reader whose role is listed gains no rights they did not already have.

7.3 Licences travel with the recipe

Each recipe carries a licence, and the four reference recipes deliberately carry different ones — CC BY 4.0, CC BY-NC 4.0 and Open Data Commons ODC-BY — to make the point that a licence is a property of the individual recipe. A non-commercial licence on a recipe that a consultancy wishes to apply on a municipality's behalf is a real constraint, and the interface surfaces it at the point of discovery rather than burying it.

7.4 Ethics and the Governance & Ethical Board

D3.6 assesses the ethical implications of the three analytical models. Because a recipe binds a capability rather than a named model, and because the consuming city chooses which registered engine satisfies that capability, the ethical assessment cannot be attached to the recipe once and forgotten. Two obligations follow: a recipe must carry forward any constraint that applies to every engine capable of satisfying it, and a hub registering an engine must attach that engine's own assessment. The publication gate described in section 6.2 is where the Governance & Ethical Board of D2.3 exercises this check.

The wider instruments — the IAMA, the EU AI Act, the Data Governance Act and the Algoritmeregister — bear on recipes that bind models producing decisions affecting people. A recipe that identifies vulnerable infrastructure ahead of a design intervention is closer to that territory than one that renders a heat map, and the Board's assessment should record which.

7.5 Logging

The mockup performs no logging, since it performs no execution. A production Cookbook should record who resolved a configuration from which version of which recipe, which resources and engine were bound, which mappings were applied, and what the resulting run produced. This is the provenance loop shown returning to the catalogue in Figure 1, and it is what makes an analysis reproducible and auditable after the fact.

8 Replication across the three pilot cities

8.1 Author once, consume three times

The stormwater recipe was authored in Rotterdam against Rotterdam's data and Rotterdam's licensed engine. The replication claim is that 's-Hertogenbosch and Tampere can apply it without editing it. Three mechanisms make that true.

1. **Capability indirection.** The recipe requires a flood simulation engine, not Rana. Each city binds whichever registered engine satisfies the capability and the execution pattern.
2. **Format-level portability.** The recipe requires formats, not named datasets. Figure 7 shows all three city base datasets satisfying its format requirements before any city-specific work is done.
3. **Locally resolved semantics.** Attribute-level differences are absorbed by the mapping step, using PPB mappings where they exist and locally configured ones where they do not.

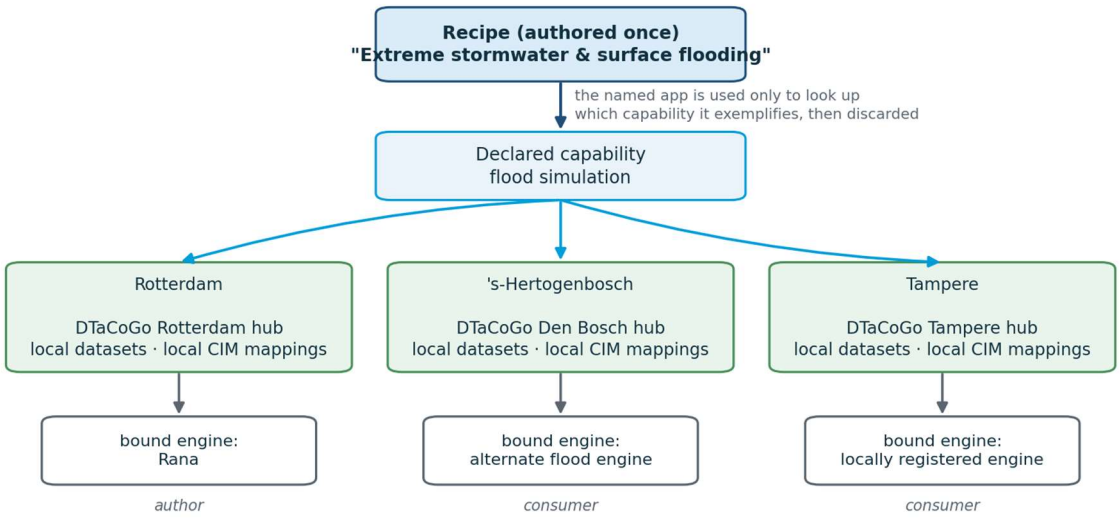


Figure 25 — One recipe, three cities, three engine bindings. The recipe is not edited.

8.2 What changes, and what does not

Element	Travels unchanged	Resolved per city
Required input roles and formats	Yes	—
Parameter contract	Yes	—
Execution pattern	Yes	—
Output formats	Yes	—
Bound datasets	No	Selected from the consuming hub's own catalogue
Bound engine	No	Any registered engine offering the required capability
Attribute mappings	Partly	PPB mapping where one exists; otherwise configured locally

Element	Travels unchanged	Resolved per city
Viewer	No	Chosen against local GPU and budget constraints
Cost	No	Depends on ownership, partners and licences in the consuming hub

Table 8 — What is portable and what is resolved locally.

8.3 The walkthrough as a worked example

The configuration illustrated throughout chapter 4 is a complete worked example in the Rotterdam hub. The terrain model and building footprints were bound to owned Rotterdam resources at no cost. The surface permeability layer was absent, so a sourcing partner was engaged at €1,200, leaving that requirement pending. Rana was bound as the engine at €500. One mapping resolved through PPB; the other was configured by hand and left incomplete. The Unreal Viewer was selected at €200. The result: a total of €1,950, one item blocking the wizard, two blocking deployment, and a precise statement of what must happen before the recipe can run.

That is the Cookbook working as intended. The configuration is not finished, and the interface says so specifically rather than either pretending otherwise or refusing to proceed.

9 Maturity, gaps and roadmap

9.1 What the mockup demonstrates

Table 9 states the maturity of each aspect at project closure, on five levels. **Demonstrated in mockup**: shown in the public mockup against fixture data and exercised by its designers; not implemented operationally and not tested with municipal users. **Operational in pilot**: working in the pilot environment outside the mockup, with evidence in an earlier deliverable. **Tested with end users**: confirmed with planners in the pilot cities; no aspect had reached this level by project closure. **Future work**: not demonstrated; on the roadmap in Table 10. **Defective**: a specific gap identified in the mockup or in the published contract, with its status at closure given in the evidence column and in Table 10.

Aspect	Maturity at project closure	Evidence
Guided discovery for users who do not know recipes exist	Demonstrated in mockup	Figure 3; live match count across four questions
Catalogue and faceted search over recipe metadata	Demonstrated in mockup	Figure 4; nine facets over four reference recipes
Constraining shared catalogues rather than duplicating them	Demonstrated in mockup	Figures 7 and 14; locked format facets
Four-step configuration with a genuine dependency order	Demonstrated in mockup	Figures 12 to 19
Two-severity readiness with precise attribution	Demonstrated in mockup	Figures 20 to 22
Capability indirection over named applications	Demonstrated in mockup	Figure 16; two engines offered for one requirement
Surfacing PPB predefined mappings in configuration	Demonstrated in mockup	Figure 18; mapping attributed to PPB. The mappings themselves work in the PPB platform; how to register them in the OUP is still open
Transparent cost accumulation including owned resources	Demonstrated in mockup	Table 4; €1,950 total
Governance metadata as advisory signal	Demonstrated in mockup	Owner hub, findability and licence on every record
Semi-automated data delivery to the three models via the CIM and the OUP	Operational in pilot	D3.1 and D3.2; Rana, Tygron and dat.Mobility all passed. Underpins step 1 and step 3 of the wizard
Scenario parameters set in the Unreal Digital Twin application	Operational in pilot	Chapter 5; models registered as APIs in the OUP hubs and called through a proxy on behalf of the logged-in user
Live catalogue federation via OGC API — Records	Future work	Static fixtures only; mapping in Table 7 is indicative. The mockup serves to identify the additions to the standards that federation requires
Execution through a Cook orchestrator	Future work	No execution runtime exists in the project
Fit of the slot model to how municipal users work	Future work	Exercised by its designers only (section 9.2); user testing planned with the first replication city, Groningen
Parameter bounds in the published contract	Defective	Section 5.3; *min*, *max* and *step* omitted. Unresolved at closure; in progress in OGC API — Processes, then to be implemented in the Unreal Digital Twin application
Cross-dataset attribute satisfaction	Defective	Section 4.4; *infiltration_rate* sought only in the mapped dataset. Open at closure; concerns attribute values as well as names; demonstration only, needs further investigation; PPB services are the interim solution
Tab naming matched to tab content	Defective	Section 3.4.1; Development tab holds provider onboarding. Rename to Provider Onboarding decided; not yet applied to the mockup

Table 9 — Maturity of each aspect of the Cookbook concept at project closure.

9.2 What it does not establish

Three limits should be stated without hedging. The mockup performs **no live execution**, so nothing here demonstrates that a resolved configuration would actually run. It uses **fixtures rather than a federated catalogue**, so the scaling behaviour of faceted search over thousands of records is untested. And it has been exercised by its designers rather than by **planners in the three cities**; the claim that the three-answer slot model matches how a municipal user thinks is a design hypothesis that user testing has not yet confirmed. No aspect of the Cookbook had therefore reached the level tested with end users by project closure, and the statuses in Table 9 should be read in that light. User testing is planned with the first replication city, Groningen.

9.3 Roadmap

Table 10 gives the status of each phase at project closure. None of the items planned for the immediate and short term was completed within the funded lifetime. User testing with planners moves to the first replication city, Groningen. The remaining work has been taken up in the Geonovum Testbed 2026 (Digital Twin as a Service, phase 2), which is to deliver a new version of the mockup and recommendations to the standards for real-world implementation.

Phase	Work	Depends on
Immediate — before any plug-in is built	Carry parameter bounds in the published record; distinguish fixed from user-settable parameters; rename the Development tab to reflect its content. Status at closure: not completed. The rename to Provider Onboarding has been decided. Parameter bounds are unresolved: work is in progress in OGC API — Processes, after which they must be implemented in the Unreal Digital Twin application. Continued in the Geonovum Testbed 2026.	Section 5.3; section 3.4.1
Short term — planned within the funded lifetime; moved to post-project work	Cross-dataset attribute satisfaction in step 3; user testing with planners in the three cities; complete the CIM mapping coverage behind step 3. Status at closure: not completed. User testing moves to the first replication city, Groningen. Cross-dataset matching, which must cover attribute values as well as names, and CIM mapping coverage, including how PPB mappings are registered in the OUP, are taken up in the Geonovum Testbed 2026; PPB services are the interim solution.	D3.1 Annex A1
Medium term — 2027, Werkplaats ecosystem	Publish recipe records against OGC API — Records; align with DCAT-AP-NL; federate the catalogue across hubs. Status: taken up in the Geonovum Testbed 2026, including recommendations for the additions to DCAT-AP-NL and OGC API — Records that recipes require.	Table 7; national alignment
Longer term	Execution through a Cook orchestrator; run provenance returning to the catalogue; recipe versioning in production use. Status: a first Cook recipe executor is part of the Geonovum Testbed 2026.	nLDT Cook; execution runtime

Table 10 — Roadmap from the current mockup towards a production Cookbook, with status at project closure.

10 Conclusions

The Cookbook is the mechanism that turns a recipe from a description into a deployment. Without it, D4.1's recipes are documentation: accurate, reusable in principle, and dependent in practice on someone in each consuming city doing the binding work by hand and leaving no record of it. With it, a recipe becomes something a city can search for, apply against its own holdings, cost, and take to a decision — with every unresolved requirement named.

Three things make the design work. Binding a **capability rather than a vendor** is what allows a recipe to cross a municipal boundary without being edited. The **two-severity readiness model** is what allows a city to plan and budget for an analysis it cannot yet run, which is the normal condition at the start of a project rather than an edge case. And **constraining the platform's own catalogues** rather than duplicating them is what keeps the Cookbook from becoming a second source of truth that drifts from the first.

The division of responsibility over scenario parameters is a deliberate boundary and, in the project's judgement, the correct one: the contract belongs to the recipe, the controls belong to the viewer, and iteration happens where the result is seen. The one defect in that arrangement — the omission of parameter bounds from the published record — is small, specific, and should be fixed before any plug-in is written against the contract. It remained open at project closure and is carried into the Geonovum Testbed 2026 (Table 10).

For OASC and the wider DS4SSCC community, the transferable result is not the mockup. It is the interaction model: the slot structure with three honest answers, the fixed dependency order of the four steps, the separation of blocked from pending, and the discipline of naming precisely what is missing. Those hold independently of the Open Urban Platform, and they are what a replication mechanism needs in order to be usable by someone who was not in the room when the recipe was written.

Annex 1 — The four reference recipes

The mockup carries four recipes, chosen to span the sectoral use cases of D2.2 and to exercise different structural features of the recipe model.

Recipe	Capability and inputs	What it exercises
Extreme Stormwater & Surface Flooding Vulnerability Workflow	Flood simulation. Digital terrain model (3dterrain), building footprints (3dtiles), surface permeability (wfs). Output: 3D Tiles, WMS. CC BY 4.0.	Mixed input formats; asynchronous execution; the full parameter block; both a PPB predefined mapping and a hand-configured one.
Urban Heat Mitigation & Tree Placement Strategy	Heat mapping. 3D building models and canopy cover. CC BY-NC 4.0.	A non-commercial licence as a real constraint on downstream use.
Development-Driven Traffic Impact & Flow Analysis	Mobility modelling. Future building capacities into a traffic engine. ODC-BY.	A third licence family; phased inputs across construction stages.
Climate-Induced Emergency Evacuation & Road Closure Routing	Chained: a flood tool connector feeding a traffic tool connector. CC BY 4.0.	Composition — the output of one capability as the input of another, and the clearest case for replication on local data.

Table 11 — The four reference recipes carried by the mockup.

Annex 2 — Screenshot register

All screenshots were captured from https://fi-group.github.io/dtacogo_recipes/ on 31 August 2026 at a viewport of 1484 × 815 pixels. Routes are relative to that base URL.

Figure	Route	Subject
3	#/	Guided search with live match count
4	#/recipes	Catalogue and the nine facets
5	#/recipes/recipe-rana/details	Recipe record: identity and discovery
6	#/recipes/recipe-rana/details	Recipe record: inputs, formats, preview
7	#/recipes/recipe-rana/data	Compatible Data with locked formats
8	#/recipes/recipe-rana/edit	Development tab
9	#/recipes/recipe-rana/edit/deployment	Deployment tab
10	#/recipes/recipe-rana/edit/user	User tab runbook
11	#/recipes/recipe-rana/edit/orchestrator	Orchestrator projection
13	#/recipes/recipe-rana/wizard	Step 1, ingredient slots
14	#/recipes/recipe-rana/wizard	Resource picker modal
15	#/recipes/recipe-rana/wizard	Sourcing partners
16	#/recipes/recipe-rana/wizard	Step 2, matching engines
17	#/recipes/recipe-rana/wizard	Attribute mapping modal
18	#/recipes/recipe-rana/wizard	Step 3, mixed mapping states
19	#/recipes/recipe-rana/wizard	Step 4, destination and viewer
20	#/recipes/recipe-rana/wizard	Blocked requirements
21	#/recipes/recipe-rana/wizard	Pending requirement
22	#/recipes/recipe-rana/status	Status tab
23	#/recipes/recipe-rana/edit/orchestrator	Parameter block

Table 12 — Screenshot register. Figures 1, 2, 12, 24 and 25 are schematic diagrams prepared for this deliverable and are not screenshots.

Annex 3 — Recipe record projection

Listing 1 reproduces the Orchestrator tab of the stormwater recipe **as it is emitted today**, abridged. Ellipses mark omitted content; nothing else has been added or altered. Listing 2 shows, separately, the addition proposed in section 5.3 — it is not present in the current output and should not be read as though it were.


```
{
  "type": "recipe",
  "conformsTo": [
    "http://www.opengis.net/spec/recipe/0.3/conf/core",
    "http://www.opengis.net/spec/recipe/0.3/conf/profiles",
    "http://www.opengis.net/spec/recipe/0.3/conf/types-schemas"
  ],
  "id": "recipe-rana",
  "title": "Extreme Stormwater & Surface Flooding Vulnerability Workflow",
  "description": "Outlines the data pipeline and rain parameters ...",
  "owner": "Digital Twin a CoGo - DS4SSCC",
  "findability": "PUBLIC",
  "license": "CC BY 4.0",
  "executionPattern": "asynchronous (JobID poll)",
  "projectPhase": [
    "Preparation",
    "Initiative"
  ],
  "targetRoles": [
    "Project Engineer",
    "Project Coordinator"
  ],
  "parameters": [
    {
      "name": "schematisation",
      "type": "dataset_reference",
      "unit": null,
      "required": true,
      "default": "input-rana-schematisation"
    },
    {
      "name": "simulation_duration",
      "type": "number",
      "unit": "hours",
      "required": true,
      "default": 24
    },
    {
      "name": "rain_duration",
      "type": "slider",
      "unit": "minutes",
      "required": true,
      "default": 60
    },
    {
      "name": "rain_intensity",
      "type": "slider",
      "unit": "mm/hr",
      "required": true,
      "default": 40
    }
  ],
  "inputDatasets": [
    {
      "role": "schematisation",
      "format": "3dtiles",
      "ref": "datasets/9008d9339a6ed970b2045e72/resources/res-rana-goudsesingel"
    }
  ],
  "compatibleApps": [ ... ]
}
```

Listing 1 — The recipe record exactly as the Orchestrator tab emits it today, abridged. Captured 31 August 2026 from #/recipes/recipe-rana/edit/orchestrator.

The addition proposed in section 5.3 concerns the two parameters typed as sliders. Bounds are what a viewer plug-in needs in order to render a control, and they are the only fields missing from an otherwise complete contract:

```
{
  "name": "rain_intensity",
  "type": "slider",
  "unit": "mm/hr",
  "required": true,
  "default": 40,
  "min": 0,              // proposed — not currently emitted
  "max": 120,            // proposed — not currently emitted
  "step": 5,             // proposed — not currently emitted
  "settableAtRunTime": true   // proposed — not currently emitted
}
```

Listing 2 — Proposed extension of a slider parameter. The bound values shown are illustrative; the recipe author sets them.

About the European Data Space for Smart Communities

The European Data Space for Smart Communities (DS4SSCC-DEP) initiative is a pivotal deployment following the preparatory action for a Data Space for Sustainable and Smart Cities and Communities (DS4SSCC). Emphasising sustainability aspects and diversity in communities, DS4SSCC developed a multi-stakeholder data governance scheme, created a blueprint for the European DS4SSCC, delivered priority datasets, and developed a roadmap towards a mature DS4SSCC.

DS4SSCC-DEP's vision revolves around creating a well-governed federated territorial, place-based data space for smart communities available for developers and infrastructure providers, aligning with the prospects outlined in Europe's Digital Decade objectives. This approach distinguishes itself from sectoral data spaces by encompassing diverse domains, and underpinning governance across all levels of society. The key objectives of the project are:

- **Objective 1: Establishing a Federated and Innovative Data Space**. This phase focuses on creating a large-scale data space controlled by public data holders, ensuring alignment with the Smart Middleware Platform and broader data space ecosystems. It also aims to offer middleware service solutions facilitating data sharing and management while refining the blueprint based on gained experience.
- **Objective 2: Pilots and Refinement of the Data Space Blueprint**. Through 10-12 cross-sectoral data pilots in various EU communities, this objective aims to validate and refine the data space blueprint. Using common high-value data sets, it seeks to create added value by combining data from various domains, contributing to the sustainability plan, and minimising infrastructure investment.
- **Objective 3: Fostering Innovation and Ecosystem Alignment**. Engaging stakeholders, compliance with sector-specific legislation, establishing links with Horizon Europe missions, creating services available via trusted application catalogues, and enhancing ethical considerations and AI-enabled local solutions constitute the focus of this objective.

DS4SSCC-DEP, building upon established European networks, aims to significantly contribute to the sustainability goals of European citizens. It emphasises co-creation, standardisation, business models, strategies for data spaces, and envisions a federated platform.

DS4SSCC-DEP consortium includes the following partners: