



D6.4 Implementation of the inventory of innovations, related interactive tools

Work Package 6

IPB



Document Identification

Project Acronym	SMARTCHAIN
Project Full Title	Towards Innovation - driven and smart solutions in short food supply chains
Project ID	773785
Starting Date	01.09.2018
Duration	36 months
H2020 Call ID & Topic	SFS-34-2017 - Innovative agri-food chains: unlocking the potential for competitiveness and sustainability
Project Website	http://www.smartchain-h2020.eu/
Project Coordinator	University of Hohenheim (UHOH)
Work Package No. & Title	WP6 Innovation platform
Work Package Leader	ISEKI-Food Association (IFA)
Deliverable No. & Title	D6.4 Implementation of the inventory of innovations, related interactive tools
Responsible Partner	Institute of Physics Belgrade (IPB)
Author (s)	Dušan Vudragović, Petar Jovanović, Antun Balaž
Review & Edit	Katherine Flynn, Foteini Chrysanthopoulou, Gunter Greil (IFA)
Type	Report
Dissemination Level	PU – Public
Date	26.02.2020
Version	1.0 Dušan Vudragović (IPB)
	1.1 Javier Casado Hebrard (UHOH)
	1.2 Katherine Flynn, Foteini Chrysanthopoulou, Gunter Greil (IFA)
Status	Final

Executive Summary

The focus of this deliverable, as defined in the project's Description of Action [1], is to present the implementation of the inventory of innovations and related interactive tools. Our implementation followed the functional requirements and the initial design of the system presented in the deliverable D6.2 [2]. The current document describes the development stage of the system and its components in the middle of the project lifetime (M18) and will be updated at the end of the project (M35).

In this document we have outlined the main technical details and described the implementation of components that are structured into three layers: the frontend, the backend, and the underlying infrastructure. For each component, we have reported its function and implementation details. The significant number of components are based on existing widely used open source solutions, such as Elasticsearch [3], [4], Apache Tika [5], [6], LevelDB [7], and Casbin [8]. For the project purposes, these are made use of by developing a set of related interactive tools: REST API, document store, and dynamic platform. Together, all these components produce a workflow that realizes the SMARTCHAIN inventory of innovations.

This workflow is exposed for the frontend components through the REST API to consume the service. One of the frontend components is the dynamic platform that allows querying of the stored innovations, and the same feature will be incorporated into the SMARTCHAIN Innovation platform. Contrary to the innovation platform, whose interface is designed to support a wide range of communities outside of the project, the dynamic platform is mainly towards the project participants. Therefore, the innovation platform sends read-only requests to the innovation inventory, while the dynamic platform also supports write requests, i.e., project's hub managers and WP leaders are able to store new and edit existing information within the inventory through the dynamic platform.

After a brief introduction in Section 1, Section 2 of this deliverable gives an overview of the system's architecture, describes functions of components, and lists open source solutions that are used for the implementation of the inventory of innovations. Section 3 gives technical details on related interactive tools, components that are developed within the framework of the project and whose main purpose is to orchestrate processes and to enable interaction between different components in the system. An initial set of innovations that are stored within the database is collected in the process that is described in Section 4. Section 5 presents the deliverable conclusions, while Appendix A specifies the REST API in a more technical manner, and Appendix B contains an innovation description template, a minimal set of metadata we have used for the description of an innovation.

Table of Contents

Document Identification	4
1. Introduction	5
2. Implementation of the inventory	6
3. Related interactive tools	8
3.1 Document Store.....	8
3.2 REST API	9
3.3 Dynamic platform	11
4. Initial data collection	13
5. Conclusions.....	14
A. REST API specification	15
A.1 Documents.....	15
A.2 Attachments.....	17
A.3 Search	18
A.4 Administrative	19
B. Innovation Description Template.....	20
References	22

Glossary

API	Application program interface
IDT	Innovation description template
IP	Intellectual property
JSON	JavaScript object notation
PDF	Portable document format
REST	Representational state transfer
SFSC	Short food supply chains
WP	Work package

1. Introduction

The development within the WP6 relies on the demands identified during the project preparation stage, requirements collected from actors and stakeholders at the multi-actor workshops, and currently available technology solutions. Due to the increasingly varied nature and practice of short supply chains, dependencies on different geographic conditions (culture, climate, resources, governing structures, available infrastructure, market, etc.), the consortium is primarily focused on 18 preselected case studies, existing short food supply chains, from 9 countries (2 case studies per country). During the analysis of these case studies, the project identified innovative and practical solutions relevant to the short food supply chain scale up. In order to enable knowledge transfer, innovation, and cooperation between the involved stakeholders of the studied short food supply chains, all these practical solutions are collected within a virtual environment developed for the acceleration of knowledge transfer, innovation, and cooperation - the SMARTCHAIN inventory of innovations.

The inventory of innovations allows storing, generating, sharing and utilizing information on innovations, facilitating communication between the innovation hubs. The front-end of the inventory is an interactive online portal (SMARTCHAIN Innovation platform, <https://www.smartchain-platform.eu/>) oriented towards all the stakeholders and actors, and the dynamic platform (<https://scinno.ipb.ac.rs/>), oriented to hub managers and WP leaders. The backend is the inventory (database) of available innovations, solutions, and recommendations. The development of the SMARTCHAIN Innovation platform is done as a part of task WP6.1, while its backend (inventory) and related interactive tools, such as the dynamic platform, which is the focus of this deliverable, within the task WP6.2.

The innovation inventory is implemented as a document organization and retrieval system, which supports quick finding and discovery of information related to short food supply chains. Through it, the users are able to upload, share, and discover innovations, patents, IPs, and other materials related to food supply chains. The target user group are farmers and agricultural organizations looking to optimize their operations, as well as innovation donors, i.e., researchers, technology providers, etc., who wish to raise the visibility of their innovations within a highly interested audience.

2. Implementation of the inventory

The SMARTCHAIN innovation inventory is developed based on the functional requirements and the initial design of the system architecture documented in the deliverable D6.2 - Design of the inventory of innovations and related interactive tools. In this document, the high-level architecture of the SMARTCHAIN inventory has been structured into three layers: the frontend and the backend component, and the underlying infrastructure. This is illustrated in **Figure 1**.

The main content in the innovation inventory system is uploaded by innovation donors, project's hub managers and WP leaders, through the dynamic platform (<https://scinno.ipb.ac.rs/>). This is done using the Innovation Description Template (IDT), the online web form (or offline form) that, besides innovation descriptions, supports the entry of additional data, such as geographical location, technology readiness level, potential customers, patent information, related documentation, photos, videos, etc. The current version of the IDT is given in Appendix B. All these data are used to better gauge the relevance of the innovation to various search queries and users. Such a search request could be submitted via the dynamic platform, and the same feature will be incorporated into the innovation platform. Contrary to the innovation platform, whose interface is designed to support a wide range of communities outside of the project, the dynamic platform is mainly oriented towards the project participants. Therefore, the innovation platform sends read-only requests to the innovation inventory, while the dynamic platform also supports write requests, i.e., project's hub managers and WP leaders are able to store new and edit existing information within the inventory through the dynamic platform.

Both the innovation and dynamic platform requests are handled through the SMARTCHAIN REST API component. This component supports standard create-read-update-delete operations on documents stored in the system. Technical description of the API is given in Section 3.2.

The central component of the backend layer is the SMARTCHAIN document store. It brings together all backend services and databases, in particular, technology database, indexing and search engine, metadata store, and document analyzer. In this layer, information is organized into JSON document structures, which can be extended by an arbitrary number of additional fields. The system is developed to support such an extension, ensuring that new fields are searchable either via free-form queries, which do the full-text search, or structured queries, which can give more specific match criteria. The documents can also have file attachments, which can be image data, PDFs, Word documents, spreadsheets, document scans, etc. All these attachments are processed in the background by the document analyzer to extract any searchable text content from them. Each attachment is associated with a field in the document structure, where the extracted text and file metadata are stored. The attachment files themselves are stored in an integrated LevelDB database [7] that resides in the underlying infrastructure layer. In order to expose the attachments through the REST API, a corresponding unique key is assigned per attachment within the JSON document.

LevelDB was chosen as a file storage because of its simple interface and ability to be fully integrated into the document store service binary. This reduces the number of components inside the system. In essence, it is a fast key-value store that can use any binary string as either key or data. This allows us to store the attachment metadata and files without additional encoding into other formats (such as base64), which would possibly increase the size of the data. In addition to this, a direct usage of the file system would impose the need for an additional metadata store, so LevelDB proved as a more consistent approach for this.

Besides the document management, the SMARTCHAIN document search also has a support for management of system users. This is an administrative feature that is used to control the level of access to the service. Users can be created, updated, deactivated, and their access can be controlled on a REST API path basis.

Elasticsearch technology [3], [4] is used for the creation and maintenance of a search index that holds all the document structures that are put into the SMARTCHAIN Document Store. It also holds text contents from the

attached files, which are extracted by the document analyzer component. Such an index enables full-text search on any part of the document and returns matching documents ranked by how well they correspond to the search query. More specific queries, that can give more strict control on how matches are made, can be specified in the Elasticsearch query DSL syntax [3], [4]. In this syntax, a query is a JSON object, which has a number of specified fields that control matching, filtering, and paging of the results.

Since the search engine within the SMARTCHAIN Document Store can only work with text data, we use the document analyzer component to extract text information from all the files that are attached to the documents in the store. It is based on the Apache Tika library [5], [6], which can extract text content from a wide range of file formats. According to the documentation, it is very versatile as it supports over one thousand file types. The extracted text is stored in the index on corresponding documents and is included in full-text searches. The Tika service is also used to determine the mime type of attachments at upload time.

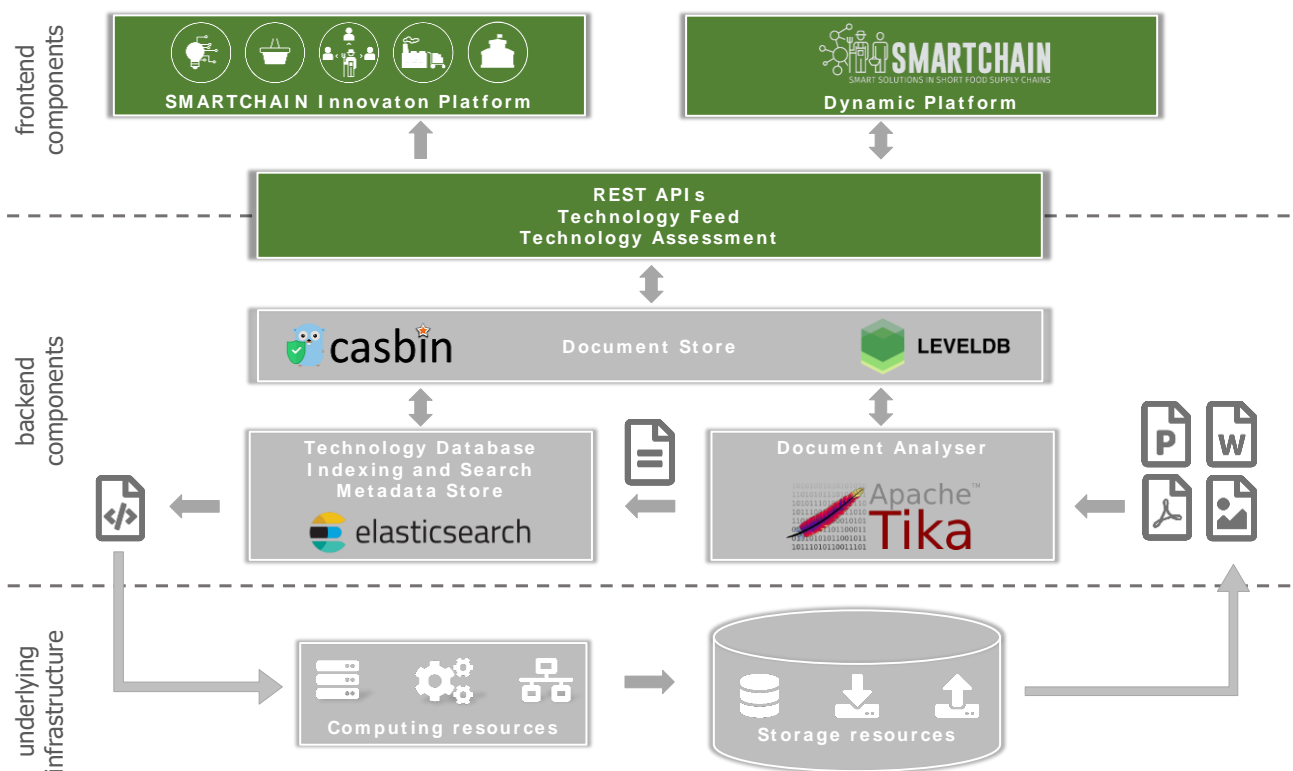


Figure 1: Architecture of the SMARTCHAIN inventory.

3. Related interactive tools

In this section, we describe interactive tools developed within the framework of the project.

3.1 Document Store

The document store is a backend service that integrates all components needed to support the SMARTCHAIN innovation platform. It is implemented in the Go programming language [9] and has the following components:

- Embedded web server that publishes the REST API;
- Component handling the search via Elasticsearch;
- Text extraction component that uses Apache Tika;
- File storage component based on LevelDB.

The entire system is compiled and statically links into a single binary, but it has a dependency on Elasticsearch and Tika services, which also depend on the Java platform [10]. The Go programming language was chosen as an implementation platform because it enables easier asynchronous programming, which was used to orchestrate all the background work that gets executed by the system in response to API requests. Another helpful feature of it was that it can make static binaries that do not require any dependencies to be installed on the deployment target.

The HTTP server from the Go standard library's NET/HTTP package was chosen for hosting the REST API. It has asynchronous processing of HTTP requests, which enables it to scale to a much larger number of connections than it would if it instantiated a thread or process per request. It is configured to support HTTPS protocol if the SSL certificate is available. The location of the server host certificate is given via the `--cert` argument and the private key is passed through the `--key` argument. The details of the hosted API are given in Section 3.2 and Appendix A.

The authorization is implemented using the Casbin library [8], which supports the PERM (Policy, Effect, Request, Matchers) metamodel for specifying authorization schemes. In our implementation, we use the authorization model based on access control lists (ACL) on API paths, that define the access level for each user role. It also has an admin superuser who can access all API calls. Anonymous users are allowed GET access on paths relevant to serving content in a read-only fashion. All requests are authenticated via the HTTP basic authentication.

The search engine is built around Elasticsearch [3], [4] which is a service wrapper around the Lucene library [11], that handles index operations, scaling, and fault tolerance. The interface it exposes is reminiscent of a database, where tables correspond to separate indices and document fields to table columns. The index used by the document store is named `sc-innovations`, and it is configured to use the edge n-gram tokenizer [4] in its default analyzer for all fields. This setting allows searching and matching of incomplete phrases in order to give meaningful results, even if there are typos in the search query.

Since the search in the SMARTCHAIN document store can only work with text data, we use Apache Tika [5], [6] to extract text information from all files that are attached to the documents in the store. Tika is a toolkit for text extraction, which can read text from more than one thousand different file types. Similar to Elasticsearch, it runs as a standalone server in the background and the document store invokes its services through an asynchronous queue.

This queue is implemented using a Go programming language channel on which each newly uploaded file is wrapped and sent in a job structure that holds its data, the document ID, and the property in the document to be updated. The job structures are consumed by worker goroutines, which are similar to threads, but are asynchronous, and in IO-bound tasks, many of which can run concurrently per single thread. The processing

in these workers invokes Apache Tika to extract any text information from the given file and to determine its mime type. The text contents and original file names are stored into the document in the index, on the specified property, which makes the information contained in the file searchable. The mime type is added as metadata into the file storage, where the original file data is also stored. This process is shown in a sequence diagram in **Figure 2**. The text extraction component depends on the Tika standalone server jar being present, and, if not, the appropriate version will be downloaded automatically and started.

The files of the attachments added to documents in the index are stored in a LevelDB storage [7]. LevelDB was chosen as a fast key-value store, which can be included in the binary of the Document Store as a static library, not to impose any new dependencies for the deployment. Along with the data from each file, we also store additional metadata including the mime type and the original file name. These metadata are used later, when serving a file to properly set response headers, so the client can render them correctly. Physically, all data are stored in the storage component of the underlying infrastructure.

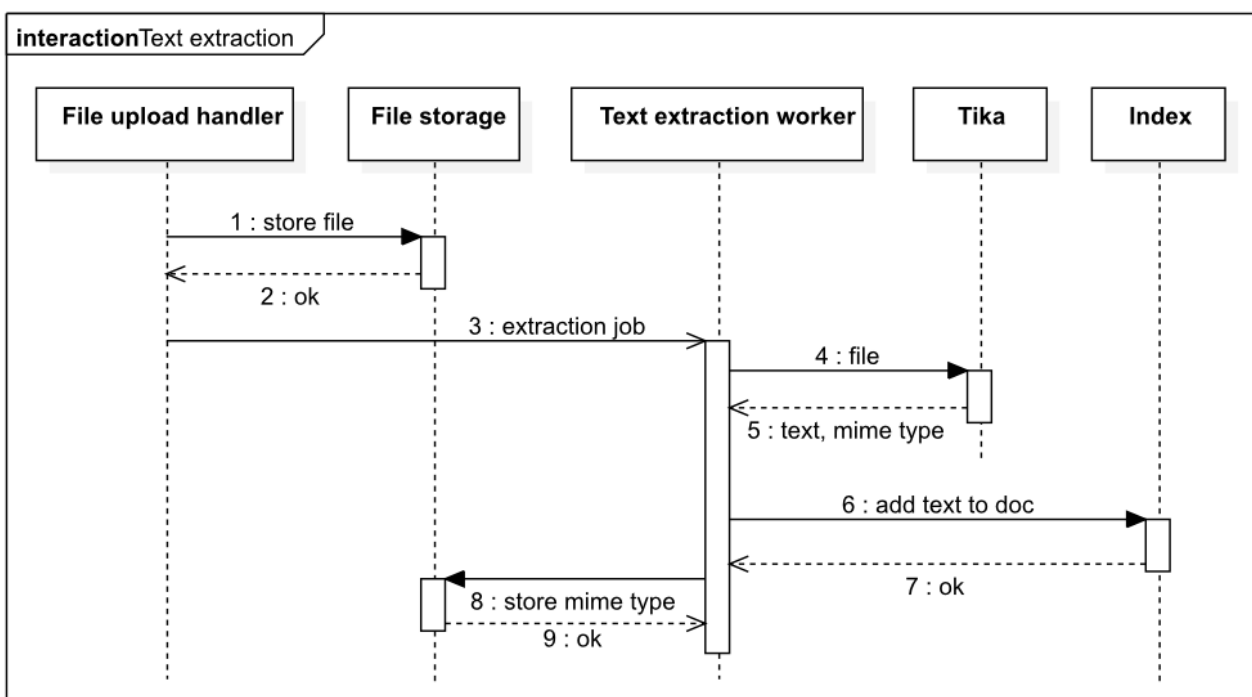


Figure 2: A sequence diagram of the file upload and text extraction process.

3.2 REST API

The REST API provides a unified interface to the document store backend. It is available on the HTTPS protocol and, depending on the configuration, could be optionally protected by the basic HTTP authentication scheme. The API exposes four types of resources: Documents, Attachments, Search, and User.

The central entity of the information schema in the document store is a **Document**. It is a JSON object with arbitrary properties, except for the ones which begin with an underscore (`_`), as they are subject to additional processing in the system. Currently, reserved properties are `_pictures` and `_documents`, which are intended for the picture and file attachments. After processing, `_pictures` and `_documents` properties are populated with arrays containing the corresponding relative URLs. Also, the original fields are populated with objects that contain file metadata and extracted text contents for full-text search. Documents are identified by `_id` field, which is assigned at document creation in the search index. The API endpoint for documents is on `/api/doc` path, and it supports CRUD (Create, Read, Update and Delete) operations on the documents using the standard HTTP verbs, as prescribed by REST:

- POST creates a document;
- GET fetches the document;
- PUT updates an existing document;
- DELETE removes a document.

A more detailed description of each operation, along with example requests and responses, is given in Appendix A.1.

The **Attachments** are files that are associated with an innovation. They are tied to a specific document on one of its properties. The property they attach to is specified during the upload, and it has structure of an array, in order to support upload of multiple files at once. Subsequent uploads to the same property are appended to the array. The attachments have two separate endpoints:

- `/api/upload` - this path accepts POST request with multipart/form-data encoding that contains the target document ID, the target property on which to place the attachments, and the attachment files;
- `/api/attachment/{key}` - this path responds to GET requests and returns the file of the attachment by the given key. The key is generated from the document ID, property, and index in the array of attachments.

When fetching an attachment, it will be served with an appropriate mime/type and original file name in response headers. Technical specification and example of the attachments' resources are given in Appendix A.2.

The underlying search engine in the document store accepts search queries specified as JSON objects. The **Search** endpoint is on `/api/search` path and it accepts POST requests through JSON query in the request body. The format of these queries is specified in Elasticsearch Query DSL, and it supports many options to control the result matching, filtering, paging, etc. More details on common queries and a request example can be found in Appendix A.3. Search results are served in an abbreviated form in a JSON array. Each element contains the following fields:

- `docid` - document ID;
- `title` - innovation title;
- `summary` - innovation description, shortened;
- `pictures` - the array of URLs of the attached pictures from the `_pictures` property.

The results are sorted by relevance score, and the main intended use for these data are to be rendered on the search results page and to provide enough information to link to the full document behind the match.

Administrative operations on the document store service include system **User** administration. These are the users who can access the REST API, and their access is controlled by ACL's rules on API paths. The authorization rules are configured in the `auth_model.conf` and `policy.csv` files, which are outside the scope of the REST API. A user entity contains the following fields:

- `username` - unique username;
- `password` - only filled out on user creation or update, otherwise blank;
- `role` - role name that controls the authorization of the user;
- `mail` - mail address;
- `active` - boolean that specifies if the user is enabled.

The requests available for the users' resource are:

- POST on `/api/users` creates a new user. The password in the request is expected to be in clear text and it will be stored hashed on the backend. It will never be sent in a response to any other request, it is only used for authenticating requests.
- GET on `/api/users` fetches the list of all system users, with password fields left blank.
- GET on `/api/users/{username}` fetches a specific user, also without a password.
- PUT on `api/users` updates a given user to the field values specified in a JSON object in the request body. Updates are total, if any field is left blank or omitted, it will be cleared in the database as well.

More details and example requests and responses are given in Appendix A.4.

3.3 Dynamic platform

The dynamic platform (<https://scinno.ipb.ac.rs/>) is a web frontend that enables read/write access to the Document Store for the SMARTCHAIN hub managers and WP leaders. It is implemented in C++ using the Wt web toolkit. The application provides a view to search and show results from the document store, and a detailed view for specific documents that can be edited there. The dynamic platform search page is illustrated in **Figure 3**, and a document page in **Figure 4**.

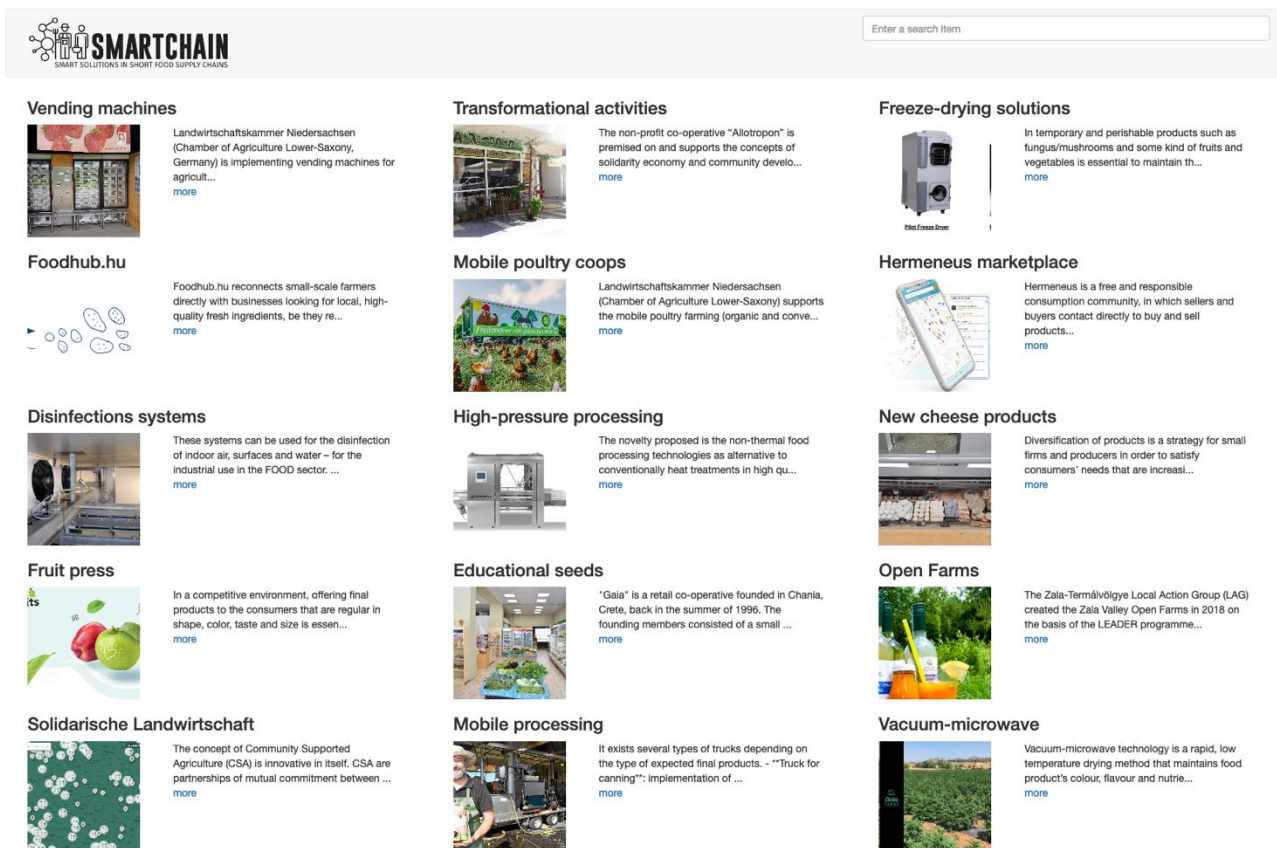


Figure 3: The dynamic platform search page.

Vacuum-microwave technology

Summary

Vacuum-microwave technology is a rapid, low temperature drying method that maintains food product's colour, flavour and nutrients. This technology enables uniform drying with flexible moisture content.

Practical benefits of this technology are: (1) **high speed processing** since drying is rapid and with significant time savings over other drying technologies (e.g. air drying and freeze drying); (2) **scalability** since machine can scale from research and design level, batch production to continuous commercial production; (3) **flexible moisture content** in final product. Namely, uniform volumetric drying allows control over final moisture content and texture achieving shelf stable final products in parallel. (4) **continuous processing** because this technology can integrate into continuous production line (5) **reduced energy saving** due to rapid drying time (6) **new product opportunities** due to a new technology that enable production of food products unachievable with other technology.

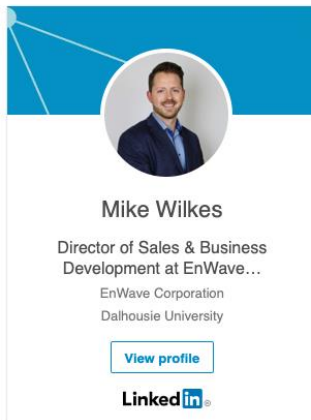
After drying process the new food products can be easily and regularly distributed in ambient temperature all over the world food market. The reason is reduced moisture and prolonged shelf life.

Type	End users	Countries
Technological innovation	Food manufacturers deal with fruits, vegetables, dairy products, ready to eat meals and snacks, meat and sea food	Worldwide

External links

<https://www.enwave.net/>

Support



Mike Wilkes
 Director of Sales & Business Development at EnWave...
 EnWave Corporation
 Dalhousie University
[View profile](#)
 LinkedIn

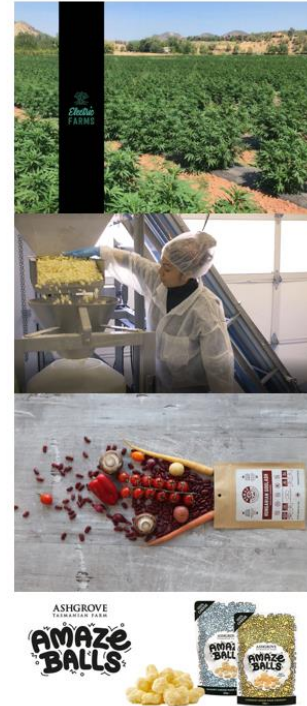


Figure 4: The dynamic platform document page.

4. Initial data collection

The initial data for the Innovation database was collected using a questionnaire filled out by the SMARTCHAIN hub managers and WP leaders. The questionnaire (given in Appendix B) contains a minimum set of information that describes an innovation, in particular:

- title of the innovation;
- picture that visually describes the innovation;
- type of innovation, whether it is technological, social, environmental, etc.;
- end users who might be interested in the innovation;
- short summary for practitioners;
- website for additional information;
- countries of origin;
- documents and publications;
- technology readiness level;
- people involved, who are able to provide additional information and support.

Besides this basic set of metadata, additional information is supplied in the form of documents, papers, external websites, etc. At the moment, the survey collected over 18 descriptions of innovations ranging from agriculture monitoring drones to new types of cheese. All this information was indexed and stored in the SMARTCHAIN document store.

Additional data is expected to be incorporated in the dynamic platform soon based on the more than 110 descriptions of innovations prepared by WP2 (Tech- and non-technological innovations).

5. Conclusions

As it is described in the SMARTCHAIN DoA, the project aims to foster and accelerate the shift towards collaborative short food supply chains and to introduce new robust business models and innovative practical solutions that enhance the competitiveness and sustainability of the European agri-food system. This is realized by the analysis of the technological and non-technological (WP2), social (WP3), consumer (WP4), environmental (WP5), and business and policy (WP7) specific factors related to short food supply chains, which will result in identification of the key parameters that influence sustainable food production and rural development.

The WP6 supports these activities by developing a virtual environment for knowledge transfer, innovation, and cooperation for all the stakeholders of the short food supply chain. From the beginning of the project, we have expected a lot of unstructured information to be stored within the inventory system. Therefore, to enable an efficient search for large quantities of unstructured data, we have created the index that holds searchable information extracted from documents or manually uploaded to the system. It is organized to support full-text search on every available piece of information. We have identified available open source solutions that could be reused for project purposes and developed a set of related tools that orchestrate the workflow in a seamless manner.

In this document, we have reported technical details of the frontend and backend components. The central part of the system is the document store that integrates all components within the SMARTCHAIN inventory of innovations, and the REST API that allows frontend components to consume the service. In addition to these, we have documented the dynamic platform, which demonstrates usage of the provided REST API, and the initial database population.

A. REST API specification

This appendix gives a more technical specification of the developed SMARTCHAIN REST API. Up to date version of this specification can be found at <https://scinno.ipb.ac.rs/api.html>.

A.1 Documents

POST /api/doc Add document to db	GET /api/doc/{id} Fetch specific document
<p>Example URI</p> <p>POST /api/doc</p> <p>Request Hide</p> <p>Headers</p> <pre>Content-Type: application/json</pre> <p>Body</p> <pre>{ "title": "Vending machines for agricultural fresh food products", "type": "Technological innovation", "endusers": ["farmers", "gardeners", "food producers"], "summary": "Landwirtschaftskammer Niedersachsen (Chamber of Agriculture L", "website": ["http://www.agridee.ch/"], "country": ["Switzerland", "Germany"], "documentsAndPublications": ["https://www.youtube.com/watch?v=lZxgEmkWSNA"], "involvedPeople": [{ "name": "Thomas Stuber", "mail": "info@agridee.ch", "url": "https://ch.linkedin.com/in/thomas-stuber-563094156" }, { "name": "Nathalie Stuber", "mail": "nathalie.stuber@agridee.ch" }], "pictures": [] }</pre> <p>Response 200 Hide</p> <p>Headers</p> <pre>Content-Type: application/json</pre> <p>Body</p> <pre>{ '_id': '8WZ5EHABGgCqiXLaOYp0', '_index': 'sc_innovations', '_primary_term': 2, '_seq_no': 5, '_shards': {'failed': 0, 'successful': 1, 'total': 2}, '_type': '_doc', '_version': 1, 'result': 'created' }</pre>	<p>If the document contains <code>_pictures</code> and <code>_documents</code> arrays which are not empty, arrays at properties <code>pictures</code> and <code>documents</code> will contain string urls for respective picture or document attachments.</p> <p>Example URI</p> <p>GET /api/doc/SXS0zm0Bafn51MDMJSM1</p> <p>URI Parameters Hide</p> <p><code>id</code> <code>string (required)</code> Example: SXS0zm0Bafn51MDMJSM1 id of a document</p> <p>Response 200 Hide</p> <p>Headers</p> <pre>Content-Type: application/json</pre> <p>Body</p> <pre>{ "_documents": [], "_pictures": [], "country": ["Switzerland", "Germany"], "documents": [], "documentsAndPublications": ["https://www.youtube.com/watch?v=lZxgEmkWSNA"], "endusers": ["farmers", "gardeners", "food producers"], "involvedPeople": [{ "mail": "info@agridee.ch", "name": "Thomas Stuber", "url": "https://ch.linkedin.com/in/thomas-stuber-563094156" }, { "mail": "nathalie.stuber@agridee.ch", "name": "Nathalie Stuber" }], "pictures": null, "summary": "Landwirtschaftskammer Niedersachsen (Chamber of Agriculture L", "title": "Vending machines for agricultural fresh food products", "type": "Technological innovation", "website": ["http://www.agridee.ch/"] }</pre>

<p>PUT /api/doc/{id} Update a document</p> <p>Updates the document specified by <code>id</code> and sets only the fields from the request. In the example below, <code>test_field</code> is set to <code>test_value</code> in the document with <code>SXS0zm0Bafn51MDMJSM1</code>. Other existing fields are left untouched.</p> <p>Example URI</p> <p>PUT /api/doc/SXS0zm0Bafn51MDMJSM1</p> <p>URI Parameters Hide</p> <p><code>id</code> <code>string</code> (required) Example: SXS0zm0Bafn51MDMJSM1 id of a document</p> <p>Request Hide</p> <p>Headers</p> <p><code>Content-Type: application/json</code></p> <p>Body</p> <pre>{ "test_field": "test_value" }</pre> <p>Response <code>200</code> Hide</p> <p>Headers</p> <p><code>Content-Type: application/json</code></p> <p>Body</p> <pre>{ "_id": "SXS0zm0Bafn51MDMJSM1", "_index": "sc_innovations", "_primary_term": 2, "_seq_no": 6, "_shards": { "failed": 0, "successful": 1, "total": 2 }, "_type": "_doc", "_version": 2, "result": "updated" }</pre>	<p>DELETE /api/doc/{id} Delete a document</p> <p>Example URI</p> <p>DELETE /api/doc/SXS0zm0Bafn51MDMJSM1</p> <p>URI Parameters Hide</p> <p><code>id</code> <code>string</code> (required) Example: SXS0zm0Bafn51MDMJSM1 id of a document</p> <p>Response <code>200</code> Hide</p> <p>Headers</p> <p><code>Content-Type: application/json</code></p> <p>Body</p> <pre>{ "_id": "SXS0zm0Bafn51MDMJSM1", "_index": "sc_innovations", "_primary_term": 2, "_seq_no": 7, "_shards": { "failed": 0, "successful": 1, "total": 2 }, "_type": "_doc", "_version": 3, "forced_refresh": true, "result": "deleted" }</pre>
---	---

A.2 Attachments

<p>POST <code>/api/upload</code> Upload attachment</p> <p>Attachments are uploaded using multipart/form-data POST request. This is done in order to handle file uploads in the standard way that all browsers support.</p> <p>Three fields are expected:</p> <ul style="list-style-type: none"> • docid - id of the document on which the attachment is being put • property - property of the document on which the attachment will be associated • attachment - one or multiple files to be attached. <p>Example URI</p> <p>POST /api/upload</p> <p>Request Hide</p> <p>Headers</p> <pre>Content-Type: multipart/form-data</pre> <p>Body</p> <pre>-----WebKitFormBoundary8M3sSU13u151XSJm Content-Disposition: form-data; name="docid" SXS0zm0Bafn51MDMJSM1 -----WebKitFormBoundary8M3sSU13u151XSJm Content-Disposition: form-data; name="property" _pictures -----WebKitFormBoundary8M3sSU13u151XSJm Content-Disposition: form-data; name="attachment"; filename="something.jpg" Content-Type: image/jpeg data... -----WebKitFormBoundary8M3sSU13u151XSJm--</pre> <p>Response <code>200</code> Hide</p> <p>Headers</p> <pre>Content-Type: application/json</pre> <p>Body</p> <pre>{ "Status": "OK" }</pre>	<p>GET <code>/api/attachments/{key}</code> Get attachment</p> <p>The attached file is returned as file stream with appropriate mime type and original filename set in the headers.</p> <p>Example URI</p> <p>GET /api/attachments/SXS0zm0Bafn51MDMJSM1_pictures.0</p> <p>URI Parameters Hide</p> <p>key <code>string (required)</code> Example: SXS0zm0Bafn51MDMJSM1_pictures.0 key identifier of an attachment, consists of document id and property where the file is attached in the document, concatenated with dots (.)</p>
--	---

A.3 Search

POST /api/search
Search documents

Search query is sent as a JSON object conforming to the [Elasticsearch Query DSL](#).

Some useful queries are free from query that is given in the example request, and the following query for listing all documents in the database: { "query": { "match_all": {} } }.

Fields `size` and `from` are used to control paging, with `size` being the number of results to fetch per page, and `from` is the result number from which to start fetching (`size*(page-1)`).

Example URI

POST /api/search

Request Hide

Headers

Content-Type: application/json

Body

```

{
  "query": {
    "simple_query_string": {
      "query": "free form search query"
    }
  },
  "size": 10,
  "from": 0
}

```

Response 200 Hide

Headers

Content-Type: application/json

Body

```

[
  {
    "docid": "95qICnABP8alW9vmFc8E",
    "title": "Vending machines for agricultural fresh food products",
    "summary": "Landwirtschaftskammer Niedersachsen (Chamber of Agriculture)",
    "pictures": []
  },
  {
    "docid": "-ZqkCnABP8alW9vm8s9W",
    "title": "Dusn Vending machines for agricultural fresh food products",
    "summary": "Landwirtschaftskammer Niedersachsen (Chamber of Agriculture)",
    "pictures": [
      "/api/attachments/~ZqkCnABP8alW9vm8s9W._pictures.0",
      "/api/attachments/~ZqkCnABP8alW9vm8s9W._pictures.1",
      "/api/attachments/~ZqkCnABP8alW9vm8s9W._pictures.2"
    ]
  }
]

```

A.4 Administrative

<div style="background-color: #e8f5e9; padding: 5px; border: 1px solid #ccc;"> POST /api/users Create user </div> <p>Example URI POST /api/users</p> <p>Request Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div> <p>Body</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <pre>{ "Username": "exampleUser", "Password": "plain text, will be hashed in the db", "Role": "admin", "Mail": "user@example.com", "Active": true }</pre> </div> <p>Response 200 Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div> <p>Body</p> <div style="border: 1px solid #ccc; padding: 5px;"> <pre>{ "Status": "OK" }</pre> </div>	<div style="background-color: #e8f5e9; padding: 5px; border: 1px solid #ccc;"> GET /api/users List all users </div> <p>Example URI GET /api/users</p> <p>Response 200 Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div>
<div style="background-color: #fff9c4; padding: 5px; border: 1px solid #ccc;"> PUT /api/users Update user </div> <p><i>Note: updates on User are total, i.e. every field is set to the new value in the object passed in the request.</i></p> <p>Example URI PUT /api/users</p> <p>Request Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div> <p>Body</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <pre>{ "Username": "exampleUser", "Password": "new password" "Role": "admin", "Mail": "user@example.com", "Active": true }</pre> </div> <p>Response 200 Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div> <p>Body</p> <div style="border: 1px solid #ccc; padding: 5px;"> <pre>{ "Status": "OK" }</pre> </div>	<div style="background-color: #e8f5e9; padding: 5px; border: 1px solid #ccc;"> GET /api/users/{username} Fetch user </div> <p>Example URI GET /api/users/exampleUser</p> <p>URI Parameters Hide</p> <p style="margin-left: 20px;">username <input type="text" value="string (required) Example: exampleUser"/></p> <p>Response 200 Hide</p> <p>Headers</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Content-Type: application/json</div> <p>Body</p> <div style="border: 1px solid #ccc; padding: 5px;"> <pre>{ "Username": "exampleUser", "Role": "admin", "Mail": "user@example.com", "Active": true }</pre> </div>

B. Innovation Description Template

SMARTCHAIN innovation description template

Title of innovation

(mandatory field)

Insert here title of innovation.

Picture of innovation

(preferable)

Picture that visually describes innovation. Please supply link to the picture or attach it to the e-mail.

Type of innovation

(mandatory field)

Types: technological, non-technological, social, consumer, environment, business, policy, other.

End users

(mandatory field)

Who might be interested for the innovation.

Short summary for practitioners

(mandatory field)

Practice abstract, innovation/result description (1000-1500 characters, word count – no spaces).

This summary should be as interesting as possible for farmers/end-users, using a direct and easily understandable language and pointing out entrepreneurial elements that are particularly relevant for practitioners (e.g. related to cost, productivity, etc). Research-oriented aspects that do not help the understanding of the practice itself should be avoided.

The main practical recommendation(s): what would be the main added value/benefit/opportunities to the end-user if the generated innovation/knowledge is implemented? How can the practitioner make use of the innovation/result?

Practice abstract examples are available at <https://ec.europa.eu/eip/agriculture/en/find-connect/projects/short-supply-chains-knowledge-innovation-network>

Website

(preferable)

Web site for additional information.

Country

(preferable)

Country of origin.

Documents and publications

(preferable)

Please send it attached to the e-mail or link to the webpage.

Technology readiness level

(preferable)

If applicable, see https://en.wikipedia.org/wiki/Technology_readiness_level

Involved people

(mandatory field)

List of people that are able to provide additional information. Please provide: first name, last name, e-mail, LinkedIn page.

References

- [1] Project SMARTCHAIN 773785 – Annex I - Description of the action.
- [2] SMARTCHAIN D6.2 – Design of the inventory of innovations and related interactive tools.
- [3] R. Gheorghe, M. L. Hinman, and R. Russo, Elasticsearch in Action, Manning Publications (2015).
- [4] Official Elasticsearch webpage, <https://www.elastic.co/>.
- [5] J. Zitting and C. Mattmann, Tika in Action, Manning Publications (2011).
- [6] Apache Tika - a content analysis toolkit, <https://tika.apache.org/>.
- [7] LevelDB - fast key-value storage library, <https://github.com/google/leveldb>.
- [8] Casbin - authorization library, <https://casbin.org/>.
- [9] A. A. A. Donovan, B. W. Kernighan, The Go Programming Language, Addison-Wesley (2015).
- [10] S. V. Chekanov, Numeric Computation and Statistical Data Analysis on the Java Platform, Springer (2016).
- [11] O. Gospodnetic, E. Hatcher, Lucene in Action, Manning Publications (2010).