TRACE4EU

PRODECTED DESIGNATION of ORIGIN (PDO)

HALLOUMI USE CASE

SOFTWARE SPECIFICATIONS

GOLDMAN SOLUTIONS & SERVICES

version: 5
05/11/2024
Nicosia

Contents

Overview

Objective

The objective of this use case is to develop a complete and customizable solution which can provide supply chain traceability information on the production of a PDO product to any consumer in the EU.

The below specs relate to the production of HALLOUMI cheese, a PDO product of Cyprus. However, the delivered solution can be easily customized to provide traceability information for any other product in any EU region.

HALLOUMI specific information

Ingredients used in halloumi production.

1. Milk - fresh sheep or goat's milk or a mixture thereof, with or without cow's milk added,
2. Rennet (but not pig rennet)
3. Mint leaves.
4. Salt

PDO Production rules

1. If cow's milk is added in addition to sheep or goat's milk, it's proportion must not be greater than the proportion of sheep or goat's milk or the mixture thereof.
2. Sheep and goat's milk comes from local breeds and their crosses.
3. Cow's milk comes from local black and white cows.
4. All the coarse fodder in the sheep and goats' diet is locally produced.
5. Milk and halloumi production must take place and must be packaged in Cyprus.
6. The composition of the milk used must be mentioned on the label, in decreasing order of percentage.

PDO domain model

1. Milk and mint producers must possess a valid license to operate VC (Verifiable Credential) from the **Ministry of Agriculture** (MoA)
2. Transporters of milk from farmers to halloumi producers must also possess a valid license
3. Halloumi producers must also possess a valid license to operate from the ministry of agriculture. Their license is a proof that they follow the required cooking procedure and use only permitted ingredients in their allowed proportions.
4. Independent auditors can periodically inspect the actors' premises, or a halloumi package sold in the market which is labelled as PDO, and revoke an actor's license, if found not to adhere to Halloumi PDO standards.

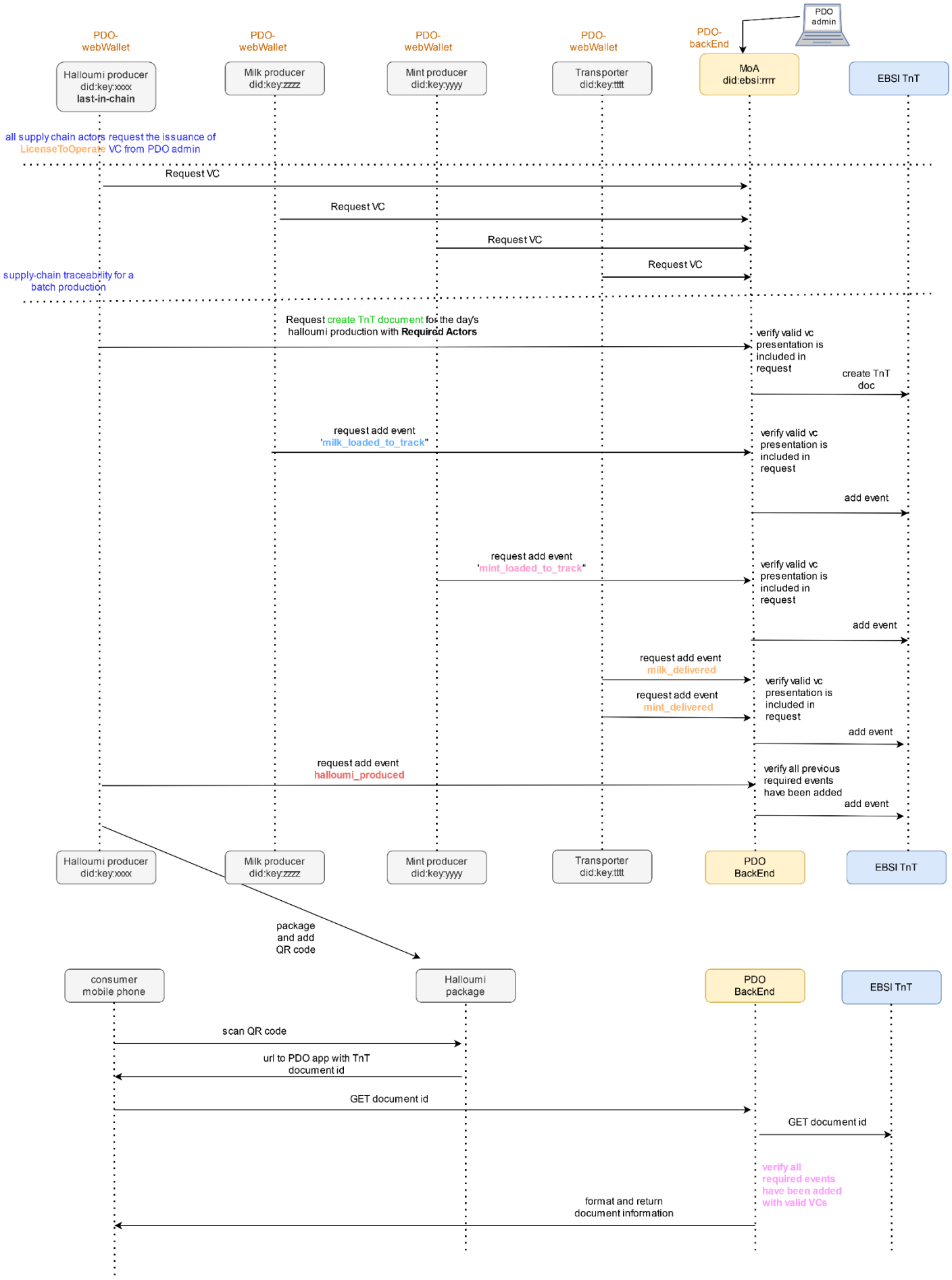
Halloumi track & Trace scenario

Assumptions

1. All actors create a **personal EBSI Web wallet** and interact with the PDO cloud application operated by MoA. The PDO's backend module interacts with the EBSI services. The PDO application is allowed to create TnT documents and issue LicenseToOperate VCs.
2. Before taking part in the supply chain, an actor has to request/get issued a LicenseToOperate VC from MoA through the PDO portal using his PDO wallet.
3. Halloumas is the owner of Halloumas Ltd halloumi production plant in Nicosia. Using his PDO wallet he requested and got issued from the MoA a license VC to produce PDO Halloumi. The license is valid for 1 year.
4. Halloumas produces 1000 packages of Halloumi each day using milk and mint from different farmers in Cyprus transported by HLM tracks to his plant. All his production is exported to various EU countries.

Daily flow -example

1. Every working day, using his PDO Wallet, Halloumas Ltd makes a request to PDO portal to create the next day's supply chain traceability document on EBSI's TnT service. PDO portal will only service this request if a VC presentation of a valid licenseToOperate VC is submitted with the request. Halloumas Ltd also specifies Alabra Farm Ltd and Papouis mint Ltd as his raw materials suppliers for the next day's Halloumi production and HLM Tracks Ltd as his trasporter. PDO creates the TnT supply chain document which contains the required event types to be added by Hallouma's specified suppliers and track operator.
2. The following day, Alabra farm loads the milk quantity requested by Halloumas to HLM's track and, using his PDO Wallet, makes a request to PDO portal to update (add event) to the supply chain document. The PDO portal will only allow the update if Alabra farm also attaches to the request a valid licence to operate VC presentation.
3. Similar to step 2, Papouis mint Ltd and HLM tracks Ltd make a request to PDO portal to update the document with the relevant events by providing their own license to operate VCs.
4. Halloumas starts production of the day's batch of 1000 halloumi packages and, following production completion, makes a request to PDO portal to update the document with details related to the specific batch production (milk proportions, expiry date, etc) and generates and attaches a QR code to each package. The QR code is a link to the PDO portal's url with the supply chain document id in TnT service.
5. The halloumi packages are exported and stored in various shops in the EU.
6. A consumer uses his mobile phone to scan the QR code on the halloumi package. The PDO portal will retrieve the TnT document, verify that all required actors have added an event to it with a valid license VC and display details of all events included in the document.


PDO admin
PDO-webWallet
PDO-webWallet
PDO-webWallet
PDO-webWallet
PDO-backEnd
Halloumi producer did:key:xxxx last-in-chain
Milk producer did:key:zzzz
Mint producer did:key:yyyy
Transporter did:key:tttt
MoA did:ebsi:rrrr
EBSI TnT
all supply chain actors request the issuance of LicenseToOperate VC from PDO admin
Request VC
Request VC
Request VC
Request VC
supply-chain traceability for a batch production
Request create TnT document for the day's halloumi production with Required Actors
verify valid vc presentation is included in request
create TnT doc
request add event 'milk_loaded_to_track"
verify valid vc presentation is included in request
add event
request add event 'mint_loaded_to_track"
verify valid vc presentation is included in request
add event
request add event milk_delivered
request add event mint_delivered
verify valid vc presentation is included in request
add event
request add event halloumi_produced
verify all previous required events have been added
add event
Halloumi producer did:key:xxxx
Milk producer did:key:zzzz
Mint producer did:key:yyyy
Transporter did:key:tttt
PDO BackEnd
EBSI TnT
package and add QR code
consumer mobile phone
Halloumi package
PDO BackEnd
EBSI TnT
scan QR code
url to PDO app with TnT document id
GET document id
GET document id
verify all required events have been added with valid VCs
format and return document information

Schemas

LicenseToOperate VC schema

The MoA admin, following an inspection to the actor's premises, can issue to the actor a LicenseToOperator VC. The VC includes the type of action that the actor can perform when taking part in the production of a PDO Product. It also specifies whether the actor is last in chain, that is, he is the producer of the final product.

Decoded jwt of the issued vc

vc: {
credentialSubject: {
id: actor-PDO-wallet-did
legalName: actor's legal name
productName: HALLOUMI/etc
allowedEvent: milk-loaded-to-track/mint-loaded-to-track/halloumi-produced/etc
lastInChain: true/false
}
}

Traceability TnT doc schema for Halloumi

The below schema is specific to halloumi traceability use case. Similar schemas can be created for any other PDO product.

TnT document –
created by PDO application following a request by Halloumi producers (lastInChain role).
in the request, Halloumi producers specify the batch-id, their vc presentation and
the wallet DIDs of their suppliers. That is, the actors that he requires to take some action so that he can complete the production of the final product.

{
from: PDO's ethereum address
documentHash: hexOf("PDO-HALLOUMI-batchId-padding"
documentMedata: Halloumi--metadata
didEbsiCreator: PDO did:ebsi
}

Halloumi-metadata

{
ceatedOnBehalfOfdid: Halloumi producer's did:key
ceatedOnBehalfOfName: Halloumi producer's name
batchid: batch-id for the day's production
requiredEvents: [
{

type: milk-loaded-to-track,
from: milk producer did:key,
fromName:
notesToActor: what is required from actor to do
},
{
type: mint-loaded-to-track,
from: mint producer did:key,
fromName:
notesToActor: what is required from actor to do
},
{
type: milk-delivered,
from: transporter did:key,
fromName:
notesToActor: what is required from actor to do
},
{
type: mint-delivered,
from: transporter did:key,
fromName:
notesToActor: what is required from actor to do
},
{
type: halloumi-produced
from: halloumi producer's did:key
fromName:
lastInChain: true
}
]
}

Event Schemas for Halloumi

An actor can add an event to the TnT document associated with a batch production, indicating that he has completed the required action assigned to him by the last in chain actor. The actor makes a request to the PDO app, providing his license VP and details of the task he performed.

Event request:
{
from: PDO app's ethereum address

documentHash: document-id created by PDO for halloumi producer
externalHash:
sender: did:ebsi of PDO app
metadata: **event-metadata**
}

milk-loaded-to-track-metadata
{

type: ‘milk-loaded-to-track’
from: actor’s did
fromName:
vcJwt: milk producer’s jwt license to operate,
eventDetials: {
milk-production-date:
milk-type:
milk-volume:
}

}

mint-loaded-to-track-metadata
{
type: ‘mint-loaded-to-track’
from: actor’s did
fromName:
vcJwt: mint producer’s jwt license to operate,

eventDetails: {
mint-production-date:
mint-weight:
}
}

milk-delivered-event-metadata
{
type: ‘milk-delivered’
from: actor’s did
fromName:
vcJwt: transporter’s jwt license to operate,
eventDetails: {
milk-delivery-date:
milk-volume:
}

}

mint-delivered-event-metadata
{
type: ‘mint-delivered’
from: actor’s did
fromName:
vcJwt: transporter’s jwt license to operate,
eventDetails: {
mint-delivery-date:
mint-volume:
}

}

halloumi-produced-event-metadata

{
  type: ‘halloumi-produced’
  from: actor’s did
  fromName:
  lastInChain:true
  vcJwt: halloumi producer’s jwt license to operate,
eventDetails: {
    production-date:
    expiry-date:
    milk-proportions:
  }
}

PDO Back-end APIs

The PDO is a cloud service operated by MoA. It exposes a number of APIs consumed by the supply chain actors' PDO wallets. It also offers an administrator's portal used by the MoA's admin and the product inspectors assigned by MoA.

The PDO is itself an enterprise wallet registered as a TI (Trusted Issuer) and which is authorized to issue LicenseToOperate VCs to the supply chain actors. It also has create access to EBSI TnT service.

Publicly exposed APIs

GET /get_license_vc?walletDID=didkey

it will return a **pre-authorized** credential offer for a **LicenseToOperate** vc. The wallet will then have to provide a pin to get the vc. Namely, following the request to /get_license_vc, the wallet will then have to call /token and /credential endpoints using the provided pin to finally get the previously issued vc.
Returns a pre-authorised credential_offer url with a signed pre-authorised_code

Example:
openid-credential-offer://?credential_offer={"credential_issuer":"http://192.168.1.6:6001/v3/tnt","credentials":[{"format":"jwt_vc","types":["VerifiableCredential","VerifiableAttestation","LicenseToOperate"],"trust_framework":{"name":"EBSI","type":"Acrreditation","id":""}}],"grants":{"urn:ietf:params:oauth:grant-type:pre-authorized_code":{"pre-authorized_code":"eyJ0eXAiOiJKV1Q.....oGDirRPZy50JcjvLEQ","user_pin_required":true}},}

POST /token

Urlencoded request:
{
 grant_type: 'urn:ietf:params:oauth:grant-type:pre-authorized_code'
 pre-authorized_code: pre-authorized_code from get_license_vc
 user_pin: the provided **pin**
}

Finds an entry in issuedvcs db with pin and walletdid (in pre-authorised_code)
Generates access_token
Updates entry in db with access_token

returns
{
 access_token:
 token_type:

id_token:
c_nonce:
}

POST /credential

Bearer: **access_token** from above

raw request:
{
types: ["VerifiableCredential",
"VerifiableAttestation",
"LicenseToOperate"],
format: ‘jwt_vc’,
proof : {
proof_type: ‘jwt’,
jwt: c_nonce signed by wallet
}
}

Validates the submitted proof and access_token, updates the issuedvc db with downloaded=true and returns the issued jwtvc from the db using the access_token.

returns
{
format: ‘jwt_vc’,
credential: jwtvc
}

GET /products?productName = name

If productName does not exists in query returns a list of product names

[HALLOUMI, KOUMANTARIA]

If productName exists returns details of the specific product. For example:
{
requiredEvents: [milk_loaded_to_track, mint_loaded_to_track, milk_delivered, mint_delivered,
halloumi_produced],
lastInChainEvent: halloumi_produced
eventsDetails: [{type:, details:[]},...]
},

GET /active_actors?product=productName

Called by a lastInChain actor to get a list of available supply chain actors for a particular product, e.g Halloumi.
Returns a list of the actors from the issuedvcs DB for the specified product name with downloaded==true && status==active && lastInChain==false. If duplicate actorDID and allowedEvent only the most recent is returned.

```
[
  {
   actorDID:
   legalName:
   allowedEvent:
   },
   {
   actorDID:
   legalName:
   allowedEvent:
   },
  ]
```

POST /init_new_batch

Called by the lastInChain actor (eg halloumi producer) to initiate a new batch production. In the request there should exist the actor's **licence to operate vp**, the batch-id and a list of the suppliers/producers that will take part in this batch production.
The PDO will create a new TnT document associated with the batch-id

Req:

```
{
 productName: eg HALLOUMI
 batchId: batch-id for the day's production (min 8 chars)
 vp_token: halloumi producer's jwt vc presentation for a valid license to operate
 presentation_submission:
  {"id":"xxx","definition_id":"pdopresentation","descriptor_map":[
    {"id":"LicenseToOperate","path":"$","format":"jwt_vp",
     "path_nested":{"id":"LicenseToOperate","format":"jwt_vc",
      "path":"$.verifiableCredential[0]"}}]
  },
requiredActions: [
 {
   type: milk-loaded-to-track,
    from: milk producer did:key,
    fromName:
    notesToActor:  what is required from actor to do
  },
 {
   type: mint-loaded-to-track,
```

```
    from: mint producer did:key,
    fromName:
    notesToActor:  what is required from actor to do
   },
   {
    type: milk-delivered,
    from: transporter did:key,
    fromName:
    notesToActor:  what is required from actor to do
    },
   {
    type: mint-delivered,
    from: transporter did:key,
    fromName:
    notesToActor:  what is required from actor to do
   },
 ]
}
```

PDO validations

Checks if all requiredActions for the specified product have been added in the request except the lastInChain.
Checks for duplicate required events
Checks if vp.vc is lastInChain and vp.vc.productName ==req.productName
For each requiredAction checks if from did is included in actor's DB and its status==active
Creates the TnT doc with id = PDO-productName-batchid-padding and the TnT metadata schema for the productName

POST /update_batch

Called by an actor in the supply chain to add an event in the TnT traceability document. The license vp, the TnT id and the new event should be included in the request

Req:

```
{
 documentId:
 vp_token:
 presentation_submission:
 eventDetails: a json that satisfies the json schema for the event the actor is allowed to.
}
```

PDO validations

If vp.vc.lastInchain check if all requiredEvents have been added first.
eventDetails should be {key1: value1, key2:value2,..} and should include all keys added using eventDetails API for the specific event type

GET /completedBatches?productName=name&actordid=did&allowedEvent=evt

Get completed batches

If BOTH actordid **AND** allowedEvent are specified it returns completed batches where the actor has performed some task. Applicable for lastInChain actors **only** who want to generate a **QR code**. It returns a list with { documentId, createdAt, batchId, createdOnBehalfOf, type }

Otherwise it returns the completed events for each completed batch containing { documentId , createdAt, batchId, createdOnBehalfOf, completedEvents }

GET /pendingBatches?productName=name&actordid=did&allowedEvent=event

Get pending batches.
If BOTH actordid AND allowedEvent are specified it returns pending batches where the actor has a pending task to perform. Otherwise it returns the pendingRequiredEvents for each pending batch.

Get all TnT docs created by PDO
Get all TnT docs where id starts with = “PDO-ProductName-“
Get all requiredEvents and events in each TnT doc as Batch

```
If actordid and allowedEvent are specified  //actor specific query
  For each batch
      If there is no event with event.lastInChain  //pending batch
          for all requiredEvents
              if exists a requiredEvent from actor and there is no event from actor
                add to list
  return list with entries { documentId, createdAt, bactId,createdOnBehalfOf,
                                                                  type,notesToActor}
else
    For each batch
    If there is no event with event.lastInChain  //pending batch
       add to list
    return list with entries {documentId, createdAt, batchId, createdOnBehalfOf,
                                          requiredEvents, pendingRequiredEvents as array of strings }
```

GET /completedTasks?productName=name&actordid=did&allowedEvent=evt

Get pending or completed batches.
ALL parameters need to be specified. Actor specific
Returns events submitted by actor for all batches containing { documentId, createdAt, batchId, createdOnBehalfOf, type, eventDetails, batchCompleted (bool) }

GET /document?id=id&fromCustomer=true/false

a consumer's device calls this API to get traceability info about a product's production batch (Halloumi) by scanning the QR code on the produc (fromCustomer=true). It is also called by actors to get the progress of the specific TnT doc (fromCustomer=false)

Gets TnT doc with id=id

If fromCustomer==**true** returns a list of the events only included in the doc in ascending chronological order. For each event, it validates the vc of the actor who added the event and adds the vc status and actor's name in the list entry. Checks that all requiredEvents have been added. Returns a **formatted web page.** Only applicable for a completed batch.

If fromCustomer==**false** It includes all the requiredEvents, the events that have been completed and the events that have not been completed (pendingRequiredEvents as Json).

GET /verifyvc

Validates a submitted jwt vc and returns its status (active, expired, revoked, invalid).
For revocation status only checks the local DB since it is assumed that all License VCs have been issuedby this backend

Admin portal APIs

All admin's portal APIs require an access token as a bearer, issued from /login API. Called from the admin portal web GUI app.

POST /login

called to get an access token. Admin's Userid and password must be provided.

GET /issue_vc

Follows an off-band request from an entity to the PDO admin to become a supply chain actor.

The json request body must contain the productName, actor's DID, the actor's legal name and the actor's allowedEvent in the supply chain (milk_loaded_to_track, halloumi_produced, etc)

Req:
{
productName: HALLOUMI/etc
actorDID: did:key:xxxx
legalName:
allowedEvent: milk_loaded_to_track/halloumi_produced/etc
}

The LicenseToOperate vc is issued and a pin is generated.
Stores in issuedvcs local DB the productName, actorDID, legalName, allowedEvent, lastInChain, status=active, pin, issuedvcjwt.

Returns a pin which must be given to the actor to be used when calling the GET /get_license_vc API.

POST /revoke_vc

Revokes a license to operate vc. The DID of the actor to be revoked must be provided.
Adds the DID in revocation list and updates the status of the actor in issuedvcs DB to "revoked".
All the entries in receivedvcs with actorDID == did are updated to "revoked".

GET /actors?productName=name

Returns a list of the actors from the issuedvcs DB for the specified product name with downloaded==true. If duplicate actorDID and allowedEvent only the most recent is returned.

```
[
  {
   actorDID:
   legalName:
   allowedEvent:
   lastInChain:
   status:
   },
   {
   actorDID:
   legalName:
   allowedEvent:
   lastInChain:
   status:
```

```
    },
  ]
```

POST /newProduct

Adds a new product for traceability in products DB.

Req:

```
{
 productName:
 requiredEvents: [event1, event2, ...]
 lastInChainEvent: event-x
}
```

PATCH /eventDetails

Adds the event details that actors must submit when updating a batch.

Req:

```
{
 productName:
 eventsDetails: [
   {type: event1, details: [key1,key2,..]},
   {type: event2, details: [key1,key2,..]}
  ]

}
```

PDO front-end (Admin portal)

A reactJs web application accessed by the PDO admin and/or product inspectors.

On loading displays login page. User enters userID/password and calls the POST /login API on PDO. The /login API returns an access token which is saved in session storage.

Menu Options displayed on a vertical menu bar on left of the page:

Issue License VC
Revoke License VC
Completed batches
Pending batches
Actors List
Products List
New Product
Event Details
Logout

Issue License VC

It is assumed the inspectors have visited the actor's premises and gave their approval for the actor's eligibility to be part of the supply chain.

The page's title is "Issue a License to Operate VC"

The page displays the following entry fields (in brown):

Product Name: display drop down list of pre-defined products (HALLOUMI/etc)
Allowed Event : display a drop down list of pre-defined events for the product name
Actor DID : the actor's wallet DID
Last In Chain : yes/no display yes if selected Allowed Event == lastInChainEvent

To get the first dropdown list call GET /products
For the second dropdown list and the lastInChainEvent, use requiredEvents and lastInChainEvent from calling GET /products?productName=name where name is the selection from Product Name entry.

On pressing "Issue VC" button the POST /issue_vc API is called with accessToken from login as bearer.

A pin is return which is displayed with the message 'please provide this pin to the actor to be used for downloading his License to operate VC"

Revoke License VC

The page displays the following entry field:

Actor DID:

On pressing the “Revoke VC” button the POST /revoke_vc API is called.

Completed batches

The page’s title is “Completed batches”

An entry field to select the Product Name is displayed

Product Name : drop down list of pre-defined products

When a product is selected it displays the list excluding the completedEvents returned from GET /completedBatches?productName=name public API

When one is selected and “**show details**” is pressed it displays completedEvents already returned from above API excluding from: DID

Pending batches

The page’s title is “Batches in Progress”

An entry field to select the Product Name is displayed

Product Name : drop down list of pre-defined products

When a product is selected display the list returned from GET /pendingBatches?productName=name API excluding requiredEvents and pendingRequiredEvents
When one is selected and “**show details**” is pressed it displays requiredEvents and PendingRequiredEvents already returned from above API excluding from: DID

Actors List

The page’s title is “Licensed actors in supply chain”

An entry field to select the Product Name is displayed

Product Name : drop down list of pre-defined products from GET /products API

On pressing “show” it displays the list returned from GET /actors?productName=name API

Products List

The page’s title is “Products with supply chain definitions”

An entry field to select the Product Name is displayed

Product Name : drop down list of pre-defined products from GET /products API

When a product is selected it displays the data returned from GET /products?productName = name

The display includes the requiredEvents, lastInChainEvent and eventsDetails.

The option to “delete” the product is given.

New Product

The page’s title is “Add supply chain definition for a new product”

An entry field to type in the product name is displayed.

On pressing the “Proceed” button the GET /products?productName = name is called and if the response is NOT [] it gives the error “already exists”.

If the response is [] the **same** page now displays the new product name and an entry field with the “Add event” button.

When the “Add event” is pressed the event is added to a list and the list is displayed below the “Add event” button.

The user can select an entry from the list and select one of the below options displayed at the end of the list:
“Delete”, “Mark as Last in Chain”, “Add product”

On “delete” the event is deleted and the displayed list is updated
On “Mark as Last in Chain” the event is marked as lastInChain and the text “Last In Chain” is displayed next to the event. Any previously marked lastInChain event is un-marked.
On “Add product” the POST /newProduct is called. Before calling the API you check if an event has been marked as lastInChain. If not the error “you must select an event as last in chain” is shown.

Event Details

The page’s title is “Add or update event details for the required event types”

Product Name : drop down list of pre-defined products from GET /products API

When a product is selected display “add or update event details” and display:

For each event type in requiredEvents display:
    event_type entry_field

and an “update” or “cancel” button at the end

Where entry_field contains the values in eventsDetails.details for the associated event_type, if any or empty.

When “update” is pressed call the PATCH /eventDetails API

PDO Wallet

A standalone reactJs web application created and used by supply chain actors.

On loading, it checks the browser’s local storage for the existence of wallet keys. If not found it offers the option to “Create a new Wallet”. If found, it displays the login page.

If create new Wallet is selected, it asks to enter/confirm a password and a new DID (did:key:xxx) and a pair of public/private keys is created and stored encrypted in local browser’s storage. The password is used to encrypt data stored in local storage.

Login Page

It displays an entry field for the password.

There is no server login. **Only local login**. When a password is selected during wallet creation the word “success!” is encrypted with the password and stored in local storage with the key= “access”. During normal login the password entered is used to decrypt the phrase stored with the key “access”. The login is successful if the decrypted word is “success!”. If successful it sets the isLoggedIn variable to true.

Following a successful login, it displays the following options on a vertical side menu bar:

My Wallet
My License VC
Req License VC
New batch
Pending Tasks
Completed Tasks
New QR code
Logout

On top left of the page, it also displays the legalName “licensed for:” productName, alllowedEvent. “Last In Chain:” yes/no retrieved from the license VC, if it exists in local storage. If more than one vc exists, it displays info from first vc and a “**select**” button. The “select” button is a drop down list with the available vcs. If a license VC is not found in local storage it displays ‘**no licensed’.**

On the page’s body it displays the list of pending requests by calling the GET /pendingBatches?productName&actordid&allowedEvent API on PDO back-end. See “Pending Tasks” option below.

My Wallet

Displays DID and private key from local storage. No mnemonic phrase is used.

My license VC

Checks if a LicenseToOperate VC exists in local storage and displays the status and info included in the VC. The status can be evaluated by calling the GET /verifyvc API on the PDO back-end. If a VC does not exist in local storage it displays:
'you have not being issued a license to operate VC yet. Please contact the PDO admin and request one'

When the selected vc is changed on top bar, the page is refreshed using the info from the new vc.

Req license VC

When the "Proceed" button is pressed it calls GET /get_license_vc API on PDO is called.
A pre-authorized offer will be returned which the wallet must use, together with a pin, to get the license VC.

If the response is a success the received jwt vc is stored in local storage **array** encrypted. It replaces any existing VC with the same productName and allowedEvent properties.

Detailed flow:

It is assumed that the PDO admin has previously issued the vc and has generated a pin code which was sent to the wallet holder.

Wallet **PDO backend**

/get_license_vc?walletDID **------ >**
< ---------- credential-offer with
pre-authorized_code //expires in 24hours

ask user to enter pin

/.well-known/openid-credential-isser --------- >
/.well-known/openid-configuration ---------- >
/token with
pre-authorized_code and PIN ---------- >
< ---------- access_token //expires in 24h. added in db rec.
//ok if record with pin && DID in
// pre-authorization_code
// exist in db

/credential with access_token and jwt proof ------ >

< ---------- vcjwt //uses access_token to get vc from db

save vcjwt in local storage array
The actor's name, productName and allowedEvent on top left of the page is updated. If there is more than one vc in the local storage the button "select" is displayed next on top horizontal bar.

access_token includes claims.authorization_details[0].types
jwt proof is a jwt signed by wallet of the c_nonce in token response

New batch

subTitle : "Create a supply chain traceability document for a new batch"

If more than one vc in local storage it uses the details from the one selected on top bar.

Checks if the license VC's role is **lastInChain**. If not, it gives an error message that 'only last in chain actors can create a new batch'. No further processing takes place.

Calls GET /products?productName = name and displays the requiredEvents from the returned list. lastInChainEvent is excluded from the list.
productName is retrieved from the current license vc.

Example:
Events Required to complete the batch production:
Event1
Event2
Event3
Event4

Displays "type the new batch id and select the supply actors to take part in the new batch. **You must select one for each required event above**"

It displays an entry field for batch ID (should be min 8 chars).

It calls GET /active_actors?product=name and displays the return list of actors (legalName, allowedEvent excluding actorDID) with a "notes to actor" entry field next to each one.
allows **multiple selection** choice.

Example:
Batch id: .....

	Actor's Name	*Allowed Event*	*Notes to actor*
...	*actorName1*	*event1*	.....
...	*actorName2*	*event2*	.....
	.....		
	.....		

When the "**Submit**" button is pressed creates a vp of the license vc and calls the POST /init_new_batch API.

When the selected vc is changed on top bar, the page is refreshed using the info from the new vc.

Pending Tasks

subTitle: "you need to take action on the following supply chain tasks"

This page is also displayed after login.

If more than one vc in local storage uses the one selected on top bar.

Displays a list of pending requests (tasks) assigned to this actor by the last-in-chain actors. Returned from the GET /pendingBatches?productName=name&actordid=did&allowedEvent=event PDO API.

Display batchId, createdAt, RequestedBy, notesToActor

RequestedBy is the createdOnBehalfOfName

The user can select **only one** of the pending events and press the button "**Mark as Complete**" at the bottom. When pressed the user will be asked to enter the event details and when the button "**Proceed**" is pressed the POST /update_batch PDO API is called.

The event details to ask are taken from GET /products?productName=name
If there is no eventDetails.details for the allowedEvent an error is displayed when "Mark as Complete" is pressed.

If success **and** the license is **lastInChain,** popup a message with "batch has been completed. Please proceed with generating a QR code for the batch items"

When the selected vc is changed on top bar, the page is refreshed using the info from the new vc.

Completed Tasks

subTitle: "List of supply chain tasks that you have completed"

If more than one vc in local storage uses the one selected on top bar.

Displays a list of completed requests (tasks) assigned to this actor by the last-in-chain actors. Returned from the GET /completedTasks?productName&actordid=myDID&allowedEvent=event PDO API. The associated batch maybe completed or pending.

Display batchId, createdAt, RequestedBy, eventDetails, batchCompleted

RequestedBy is the createdOnBehalfOfName

The user can select one from the list.

When an entry from the list is selected and the **Show** button is pressed the details of the entry are shown by calling the GET /document?id=id&fromCustomer=false API

When the selected vc is changed on top bar, the page is refreshed using the info from the new vc.

New QR Code

subTitle: "generate a QR code for a completed batch. Valid for Last In Chain actor only"

If more than one vc in local storage uses the one selected on top bar.

Checks if the license VC's role is lastInChain. If not, it gives an error message that 'only last in chain actors can generate a QR code for the final product'.

It calls the GET /completedBatches?productName=name&actorDid=mydid&allowedEvent=event and displays the returned list of batches that have been completed for which the actor is lastInChain

Display batchId, createdAt

The user can select one and press the button "**show As customer**" or "**Generate QR code**"

"show As customer" calls https://PDO-url/document?id=documentId&fromCustomer=true

"Generate QR code" a QR code is generated of the url https://PDO-url/document?id=documentId&fromCustomer=true

Display the message "please print and attach this QR code on this batch's production items"

When the selected vc is changed on top bar, the page is refreshed using the info from the new vc.

Logout

Sets the isLoggedIn variable to false and displays the login screen