Technical Documentation LACPASS

Last updated at Jan 22, 2022

Introduction

This document is presented as technical documentation for the implementation of LACPASS, detailing how the solution operates and the necessary steps that participating countries must take to join LACPASS.

The Latin American and Caribbean Vaccination Pass, LACPASS, is an application for the exchange of information on vaccination status of Latin American and Caribbean countries, which allows people who have received part or all of the COVID vaccination scheme in their country of residence, when traveling to another country in the region, they can simply and verifiably validate their vaccination status in the country of destination, without the need to carry out additional procedures such as homologation of the local vaccination certificate.

The LACPASS project is an initiative of the American Network for Cooperation in Electronic Health in Latin America and the Caribbean (RACSEL), sponsored by the Inter-American Development Bank (IDB) and executed by the National Center for Information Systems of Chile (CENS) by through the private company Create de Chile, which was awarded the tender for the development and implementation of this public good.

The technology behind LACPASS is based on the Digital Green Certificates of the European Union (EU-DGC), this repository is an open source project used in all the countries of the European Union and 24 countries outside it. This pass is multilanguage and is available in English, Spanish, French and Portuguese which are of special interest in this region. In addition, it can be digital and on paper, and has a verifiable QR code through the applications provided by the DGC. By connecting interested countries to LACPASS it is possible to use the same technology to connect to the Digital Green Certificates of the European Union.

The main objective of the LACPASS project is to connect in a secure and verifiable way the information on individual vaccination of the residents of the countries of the region in a uniform and interoperable system that facilitates travel within the region by delivering to the health and immigration authorities of the countries a tool that provides accurate and timely information on the vaccination status of passengers who are entering or passing through.

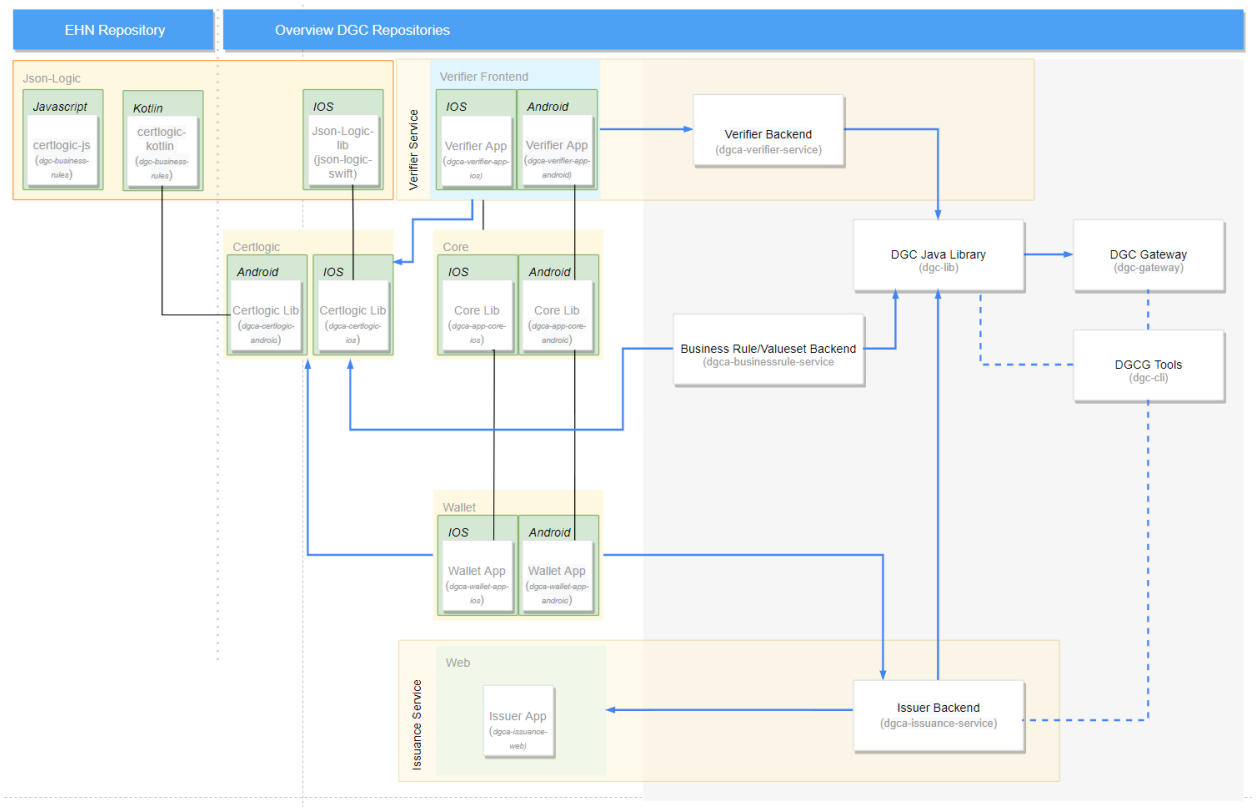
As an additional objective, it seeks to collaborate with the countries of the region so that they can connect in a simple and fluid way to the Digital Green Certificates technology of the European Union.

Architecture

As explained above, the implementation of LACPASS is based on the Digital Green Certificates projects of the European Union (DGC) and European Health Network (EHN), whose repositories can be found at the following links:

- DGC: https://github.com/eu-digital-green-certificates
- EHN: https://github.com/ehn-dcc-development

The DGC provides different repositories for the implementation of interoperability of vaccination certificates. The interaction of all these repositories is shown in the following diagram:



Functionally the repositories can be divided into 3 groups:

Logic and Synchronization

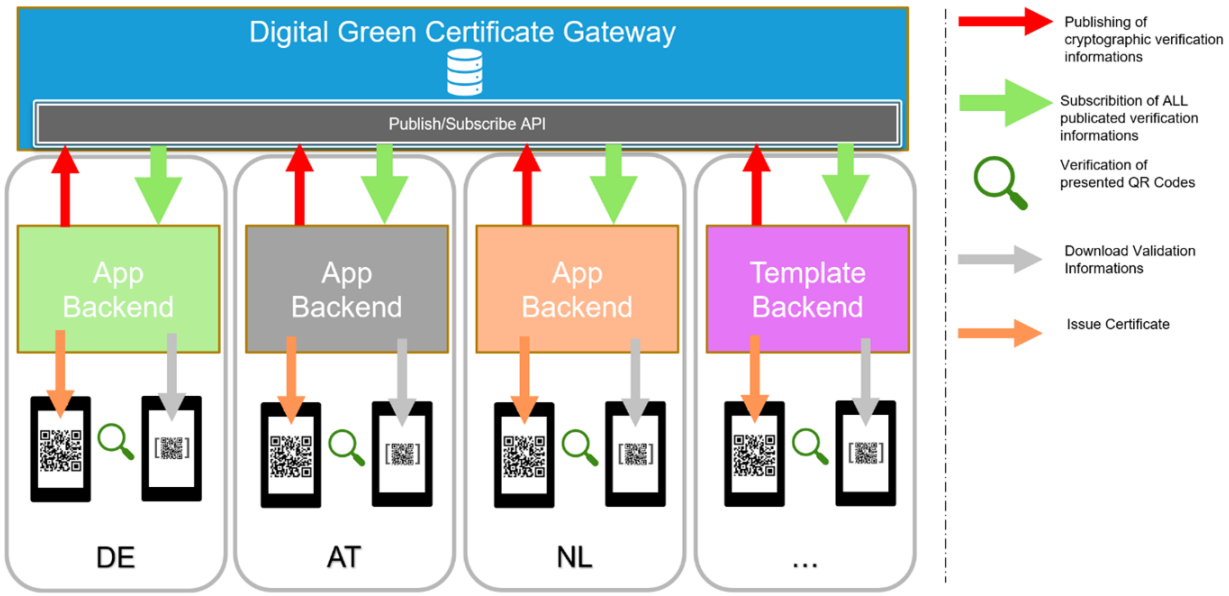
Gateway

The DGC Gateway has the purpose of serving as support for the entire DGC system, it provides all the services necessary for the secure transfer of validations and verifications between national systems. Each national system can implement its own DGC Gateway to obtain the

freedom to distribute the keys with the preferred technology and also to be able to manage national verification systems.

Additionally, if the certificate is generated in a correct standard format, any verifying device will be able to verify codes of any country that has the EU format. This works for both the verifier connected to the national system and offline systems that have the necessary public keys downloaded beforehand.

The following diagram shows the flow between the different national systems and the DGC Gateway:



Here is a link to detailed documentation of the DGC Gateway (Documentation).

Business Rule Service

The DGC Business Rule Service is one of the services connected to the DGC Gateway, this service provides the necessary rules to verify whether or not a code is valid in a national system. These rules are based on the vaccines you have, the tests performed and the recovery status of the validated person..

To generate these validation rules there is a more detailed format in this link (Business Rules Test Data).

Issuance

Issuance Service

The DGC Issuance Service is the backend system that provides both the creation and signing of new certificates (green certificates). Each country must raise this service to be able to have the certificates. In order for the certificates to be used internationally, the public keys must be shared in the Gateway so that all countries can verify the certificates. This service is used by mobile applications (Android, iOS) and by web applications.

Issuance Web

The Issuance Web is a web application that provides a user interface used to provide the necessary data in the issuance service. Certificates can also be generated in this application.

Verification

Verifier Service

To verify the certificates it is necessary to have the public keys of the appropriate national system. The DGC Verifier Service is a backend service that is used to manage the public keys obtained through the DGCG. This service is used in mobile applications to obtain public keys and verify green certificates.

To verify the certificates you can use both the verifier on **iOS** and **Android**. Both repositories contain a very simple application to scan QR codes and a verification and validation interface for these.

Security and Encryption Keys

The DGC system uses a security system based on the paradigm of public and private keys that are used to verify the authenticity of the queries and the signing of the certificates. Something important to note is that these public keys are verified directly by the DGC application and do not necessarily follow the usual HTTPS rules.

Within the repositories, different formats and standards are used for saving keys, each of these formats is described below:

- **PEM:** File containing a public key and optionally a private key in flat form. Usually only the public key is included.
- **KEY:** File containing a private key in a flat shape. This file **should never be shared** with third parties to avoid attacks and vulnerabilities.
- **P12:** File that contains a public key and optionally a private one, encrypted by a password. Normally a PEM file is taken as input to build a P12.

- **JKS:** Format similar to P12 that is able to be read by Java applications in a simple way.

Implementation

This section describes the steps that need to be taken for a new participating country to be included in LACPASS.

Technologies
The "*EU Digital Covid Certificates*" (EUDCC) repositories provide APIs that are developed in the Spring Framework using Java as the primary programming language. The databases used are Mysql and Postgresql. The certificate issuance web application is developed in React. And the mobile apps are natively developed on Kotlin (Android) and Swift (iOS). All projects except mobile apps are available through Docker.

Server Requirements
A server is required which will host the web services repositories. The characteristics of this server will depend on the estimated traffic, but a server with at least 4 vCPUs, 8 Gb of RAM and 50 Gb of disk is recommended.

A server is required which will host the web services repositories. The characteristics of this server will depend on the estimated traffic, but a server with at least 4 vCPUs, 8 Gb of RAM and 50 Gb of disk is recommended.

Pre-requirements
The steps to follow to create each of the EUDCC repositories will be given below. Pre requirements:

- OpenJDK 11
- Maven
- Authenticate with Github Packages
- Docker (optional)
- Docker-compsoe (optional)
- Node 14
- OpenSSL
- DGC-CLI

In order to install the dependencies through Maven in the repositories that use Spring as technology, you need to be authenticated by Github. For this you need to create a personal access token, which has the option "read: packages" selected. Then you must fill in the maven configuration file (in linux located in ~/.m2/settings.xml) like the one shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <interactiveMode>false</interactiveMode>
  <servers>
    <server>
      <id>dgc-github</id>
      <username>$USER</username>
      <password>$TOKEN</password>
    </server>
    <server>
      <id>ehd-github</id>
      <username>$USER</username>
      <password>$TOKEN</password>
    </server>
  </servers>
</settings>
```

Gateway

The gateway is used to share and verify information through all the countries connected to it. Therefore, it should not be included in the backend of each country, here it is explained how to set up a gateway only in order to be able to test the connection of other services. The repository can be cloned using:

```
$ git clone https://github.com/eu-digital-green-certificates/dgc-gateway
```

Keys

To run it locally you need to create a TrustAnchor. The TrustAnchor is used to sign entries in the database. To create the TrustAnchor the following command is used:

```
$ openssl req -x509 -newkey rsa:4096 -keyout key_ta.pem -out cert_ta.pem
-days 365 -nodes
```

Then the public key is exported to the Java Keystore using:

```
$ keytool -importcert -alias dgcg_trust_anchor -file cert_ta.pem -keystore
ta.jks -storepass dgcg-p4ssw0rd
```

Where "cert_ta.pem" is the public key and "dcg-p4ssw0rd" is the password of the public key. This "ta.jks" key must be placed in a folder named "certs", which must be created in the root of the repository.

Database

This repository uses a MySql database, if docker is not used to build the project, you need to install and create a base in MySql.

Configuration

To configure variables such as the directory of the public key and the connection to the database, it can be done in two ways. If Docker is used to run the project, the environment variables shown in "docker-compose.yml" can be edited. For more details on this file, the documentation is available at the following link. If docker is not used, you can edit the Spring configuration file in "~/dgc-gateway/src/main/resources/application.yml"

Execute

To build the project executable, through Maven, use the following command:

```
$ mvn clean install
```

If docker is used to run the project, an extra flag must be added to the previous command:

```
$ mvn clean install -P docker
```

This will create a "jar" file in the "~/dgc-gateway/target" directory. To run the application you use:

```
$ java -jar target/dgc-gateway-latest.jar
```

And if you use Docker, you can use:

```
$ docker-compose up --build
```

Which will upload the gateway API along with a mysql database. In order to query the API of this gateway, it is necessary to register certain certificates that belong to the backend of each country. These certificates will be from AUTHENTICATION, UPLOAD and CSCA. For this, these certificates can be created with OpenSSL:

```
# AUTHENTICATION
$ openssl req -x509 -newkey rsa:4096 -keyout key_auth.pem -out
cert_auth.pem -days 365 -nodes

# CSCA
```

```
$ openssl req -x509 -newkey rsa:4096 -keyout key_csca.pem -out
cert_csca.pem -days 365 -nodes

# UPLOAD
$ openssl req -x509 -newkey rsa:4096 -keyout key_upload.pem -out
cert_upload.pem -days 365 -nodes
```

These certificates must be signed by the TrustAnchor of the gateway (“cert_ta.pem” and “key_ta.pem”), for this the client provided by the EUDCC can be used. This can be downloaded at this link. Then using this jar, the following commands can be executed:

```
$ java -jar dgc-cli.jar ta sign -c cert_ta.pem -k key_ta.pem -i
cert_auth.pem
$ java -jar dgc-cli.jar ta sign -c cert_ta.pem -k key_ta.pem -i
cert_csca.pem
$ java -jar dgc-cli.jar ta sign -c cert_ta.pem -k key_ta.pem -i
cert_upload.pem
```

In each of these commands a "TrustAnchor Signature", "Certificate Raw Data", "Certificate Thumbprint" and "Certificate Country" will be delivered. These values have to be entered in the "trusted_party" table of the gateway database, so three new lines will be added in this table (for each of the certificates). This can be done using:

```
$ mysql --user=root --password=admin dgc
$ INSERT INTO trusted_party (created_at, country, thumbprint, raw_data, signature,
certificate_type)
SELECT
       NOW() as created_at,
       'CL' as country,
       '{Certificate_Thumbprint}' as thumbprint,
       '{Certificate_Raw_Data}' as raw_data,
       '{TrustAnchor_Signature}' as signature,
       '{AUTHENTICATION|UPLOAD|CSCA}' as certificate_type;
```

To test that the values were entered correctly, a request can be made to the gateway API using the authentication thumbprint:

```
$ curl -X GET http://localhost:8080/trustList -H "accept: application/json"
-H "X-SSL-Client-SHA256: $THUMBPRINT" -H "X-SSL-Client-DN: C=$COUNTRY"
```

Which should deliver the list of certificates in the table "trusted_parties".

Business rule

This repository contains a backend with the business rules to accept / reject the states of the COVID certificates issued by the countries. The repository can be cloned using:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-businessrule-service
```

Keys

This repository requires three keys, a trust_anchor, trust_store, and key_store. The trust_anchor is the TrustAnchor created in the gateway, the trust_store can be created using the certificate and authentication key that were registered in the gateway as follows:

```
$ openssl pkcs12 -export -in cert_auth.pem -inkey key_auth.pem -name 1 -out
tls_key_store.p12
```

The truststore is created using the authentication certificate, with the command:

```
$ openssl pkcs12 -export -in cert_auth.pem -name tls_trust -out
tls_trust_store.p12 -nokeys
```

Database

This repository uses a Postgresql database, if docker is not used to build the project, you need to install and create a Postgresql database.

Configurations

To configure variables such as the directory of the keys and the connection to the database, it can be done in two ways. If Docker is used to run the project, the environment variables shown in "docker-compose.yml" can be edited. For more details on this file, the documentation is available at the following link. If docker is not used, you can edit the Spring configuration file in "~/dgc-gateway/src/main/resources/application.yml". The most important variables are shown below:

```
# Credentials database
SPRING_DATASOURCE_URL=<CONNECTION_URL>
SPRING_DATASOURCE_USERNAME=<USER>
SPRING_DATASOURCE_PASSWORD=<PASSWORD>
```

```
# Gateway endpoint
DGC_GATEWAY_CONNECTOR_ENDPOINT=https://test-dgcg-ws.tech.ec.europa.eu

# Certificates
DGC_GATEWAY_CONNECTOR_TLSTRUSTSTORE_PATH=<PATH>
DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_ALIAS=<ALIAS>
DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PATH=<PATH>
DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PASSWORD=<PASSWORD>
DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_ALIAS=<ALIAS>
DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PATH=<PATH>
DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PASSWORD=<PASSWORD>
```

Run

To build the project executable, which is built through Maven, the following command is used:

```
$ mvn clean install
```

This will create a "jar" file in the "~/dgc-businessrule-service/target" directory. To run the application you use:

```
$ java -jar target/dgc-businessrule-service-latest.jar
```

And if you use Docker, you can use:

```
$ docker-compose up --build
```

Rules

This section briefly explains how to generate a JSON file with the certificate validation rules. These rules determine if a person who enters a country is considered suitable to enter this country, the rules are based on the vaccines administered, the tests they have performed and the state of recovery after contracting COVID. All these rules must be encoded according to the standards of the Digital COVID Certificate.

To generate the validation rules it is required to generate a json with the following:

- Valid as a CertLogic expression.
- The JSON file of every rule is validated against this JSON Schema.
- The specified AffectedFields field is checked against the fields of the DCC payload accessed from the Logic field. (DCC Schema)

CertLogic is a semantics subset that extends the JsonLogic semantics. These semantics use intuitive and simple rules to be able to verify patterns or logic within a Json file. They use logical operators such as equality ("=="), numeric operators, and so on. These operators can be found here.

Here is an example of how a Json is constructed with the CertLogic semantics:

```
{
  "<operation id>": [
     <operand 1>,
     <operand 2>,
     // ...
     <operand n>
  ]
}
```

Now to generate a file with the correct standards, the scheme must be followed correctly, for this the following fields must be added:

- **AffectedFields:** Arrangement of rules to be used from the payload (QR).
- **Country:** ISO country code. (e.g. "CL").
- **CertificateType:** Certificate type. Valid values are "General", "Test", "Vaccination", "Recovery". If, for example, the rule looks for the minimum time after a COVID test, this certificate is of the "Recovery" type.
- **Description:** Fix with the description of the rule, here all the languages that you want to support are added.
- **Engine:** Type of semantics used. (e.g. "CERTLOGIC")
- **EngineVersion:** Version of the semantics. Currently "1.2.2".
- **Identifier:** Unique identifier for the rule. It must be the pattern "^(GR|VR|TR|RR|IR)-[A-Z]{2}-\\d{4}$". For example, if the rule is "Recovery", the country is Chile and it is also the first rule, the identifier is "RR-CL-0000".
- **Logic:** Object where the rule is established. Here semantics are used to define the rule.
- **SchemaVersion:** Version of the schema used.
- **Type:** Type of the rule, it can be of acceptance (“Acceptance”) or invalidation (“Invalidation”).
- **ValidFrom:** Until what date this rule is valid (without ms and with time zone).
- **ValidTo:** From what date this rule is valid (without ms and with time zone).

- **Version:** Rule version.

To better understand how this file is generated, it will be explained in a general way how to construct the “Logic” and “AffectedFields” fields. For the field "AffectedFields" it must be understood how the payload arrives (content of the QR), the content has a standard format that can be found at this link. The payload object must contain at least one of the following fields:

- “v”: Contains everything related to vaccination (“Vaccination Entry”).
- “t”: Contains everything related to the tests performed (“Test Entry”).
- “r”: Contains everything related to recovery (“Recovery Entry”).

Each of these fields can contain specific attributes to what it represents, we will detail in the following points what each one can contain.

Vaccination Entry (“v”)

- tg: Disease or target agent.
- vp: Vaccine or prophylaxis.
- mp: Vaccine drug.
- ma: Authorized marketing company or manufacturer.
- dn: Dose Number.
- sd: Total doses (Series of doses, for example would be 2 if two doses are required).
- dt: Vaccination date.
- co: Country of vaccination.
- is: Certificate issuer.
- ci: Unique identifier of the certificate (UVCI).

Test Entry (“t”)

- tg: Disease or target agent.
- tt: Type of test.
- nm: Nucleic acid test.
- ma: Rapid antigen test name and manufacturer.
- sc: Date/Time of sample collection.
- tr: Test result..
- tc: Center in charge of the examination.
- co: Test country.
- is: Certificate issuer.
- ci: Unique identifier of the certificate (UVCI).

Recovery Entry (“r”)

- tg: Disease or target agent.
- fr: Nucleic acid test first positive date.

- co: Test Country.
- is: Certificate issuer.
- df: Date from which the exam is valid.
- du: Date until when the exam is valid.
- ci: Unique identifier of the certificate (UVCI).

There is an official document on the documentation of this standard, in this link.

To better understand how the values are chosen, we will take as an example the rule "Vaccination series must be complete (eg 1/1, 2/2)". For this example the values of "AffectedFields" would be the following:

```
"AffectedFields": [
  "v.0", // Vaccination values are required.
  "v.0.dn", // Current dose.
  "v.0.sd" // Total number of doses in the series.
]
```

Now understanding how the "AffectedFields" is assembled, following the same example, we are going to build the logic of the rule: "The vaccination schedule must be complete (for example, 1/1, 2/2)". The first thing to note is that it is a vaccination rule so the “v” part of the payload is used. In addition, as it seeks to verify the vaccination series, both "dn" and "sd" will be used where we will obtain the information of the current dose and the total of doses required respectively. Then, to validate the complete vaccination scheme, it must be verified that both values are the same, as shown in the following scheme:

```
"Logic": {
  "if": [ //  If the content is met, the rule is accepted.
    {
      "var": "payload.v.0" // Where to obtain the values is made explicit.
    },
    {
      "===": [ // The exact equality operator is used.
        {
          "var": "payload.v.0.dn" // Current Dose (Number in the series).
        },
        {
          "var": "payload.v.0.sd" // Total number of doses in the series.
        }
    }
  ]
}
```

This example was taken from the rules of Spain here. If you need more examples you can see those recommended by the EU (More examples).

Issuance

This repository contains a backend that allows the issuance of vaccination certificates. The repository can be cloned using the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-issuance-service
```

Keys

This project uses a new signing key and the previously created keys for the business rule. In fact, instead of copying the keys between the repositories, it is recommended to have a directory where all the repositories share the keys and are shared through symbolic links or docker volumes.

Signing Key

In addition to the AUTH, UPLOAD and CSCA keys generated during the gateway configuration, there is an additional key to sign the issued certificates.

This key is important to the process and it is used in several parts of the implementation:

- It is used by the issuance service to sign the requested certificates.
- The public key is uploaded to the gateway and broadcasted to the rest of the countries.
- The public key is used by the verifier service and apps to validate the issued certificates.

The signing key is generated by running the following commands

```
#SIGNING KEYS
openssl req -newkey ec:<(openssl ecparam -name prime256v1) -keyout
$COUNTRY_CODE/DSC01privkey.key -nodes -out $COUNTRY_CODE/DSC01csr.pem -utf8
-subj "/C=$COUNTRY_CODE/O=Gobierno de $COUNTRY/OU=Ministerio de
Salud/CN=Gestión de Credencial de Inmunización";

openssl x509 -req -in $COUNTRY_CODE/DSC01csr.pem -CA
$COUNTRY_CODE/cert_csca.pem -CAkey $COUNTRY_CODE/key_csca.pem
-CAcreateserial -days 730 -extensions ext -extfile dsc.conf -out
$COUNTRY_CODE/DSCcert.pem;

# SIGNER KEYSTORE
openssl pkcs12 -export -in $COUNTRY_CODE/DSCcert.pem -inkey
```

```
$COUNTRY_CODE/DSC01privkey.key -passout pass:$PASSWORD -name "firmador"
-out $COUNTRY_CODE/firmasalud.p12 -name firmasalud;

keytool -importkeystore -deststorepass ${PASSWORD} -srcstorepass
${PASSWORD} -alias firmasalud -srckeystore $COUNTRY_CODE/firmasalud.p12
-destkeystore $COUNTRY_CODE/firmasalud.jks;
```

Note that the signing key is called firmasalud.jks and some names are in spanish, change them accordingly to your language if needed.

The generated key should be included in the `ISSUANCE_KEYSTORE` variables of the docker compose configuration file.

The key should be uploaded to the gateway for its use within the verification apps. The endpoint `signerCertificate` from the gateway API is consumed to upload the key. The following commands convert the key to the format that the endpoint accepts and sign the request using the upload certificate.

```
# SIGNED DSC CMS
openssl x509 -outform der -in $COUNTRY_CODE/DSCcert.pem -out
$COUNTRY_CODE/cert.der

openssl cms -sign -nodetach -in $COUNTRY_CODE/cert.der -signer
$COUNTRY_CODE/cert_upload.pem -inkey $COUNTRY_CODE/key_upload.pem -out
$COUNTRY_CODE/signed.der -outform DER -binary

openssl base64 -in $COUNTRY_CODE/signed.der -out $COUNTRY_CODE/cms.b64 -e
-A

# UPLOAD TO GATEWAY
curl -v -X POST -H "Content-Type: application/cms" --cert
$COUNTRY_CODE/cert_auth.pem --key $COUNTRY_CODE/key_auth.pem --data
@cms.b64 http://localhost:8080/signerCertificate
```

Note that the last command uses localhost:8080 as the gateway URL but it should be changed appropriately to the actual URL.

After doing that, it is possible to check if the signing key is correctly uploaded by executing this command

```
curl -X GET http://localhost:8080/trustList/DSC -H "accept:
application/json" -H "X-SSL-Client-SHA256: $THUMBPRINT" -H
"X-SSL-Client-DN: C=$COUNTRY"
```

If the signing key was correctly uploaded, the request should respond with the data of the certificate.

Database

This repository uses a Postgresql database, if docker is not used to build the project, you need to install and create a Postgresql database.

Configurations

Like the business rule, the repository configuration is in the docker-compose.yml file, but you can also change the "src/main/resources/application.yml" file directly if you don't use docker.

The issuance service has two forms of execution: one for testing and the other connected to a gateway. Both are explained below:

Testing Configuration: This is the one that comes by default and is used to quickly test the issuance of certificates without having to install a gateway. No further configuration is required to operate in this mode and a generic test key is used to sign the certificates.

Production Configuration: This configuration connects to a gateway and allows interoperation with the other DGC services. To access this configuration you have to change the docker-compose.yml and add the following configurations to the backend

```
backend:
   environment:
      ... # KEEP WHAT IS AND ADD THE FOLLOWING
      # EMISION DE CERTIFICADOS
      - ISSUANCE_DGCIPREFIX=URN:UVCI:V1:CL
      - ISSUANCE_KEYSTOREFILE=/app/certs/CL/firmasalud.jks
      - ISSUANCE_KEYSTOREPASSWORD=dgcg-p4ssw0rd
      - ISSUANCE_CERTALIAS=firmador
      - ISSUANCE_PRIVATEKEYPASSWORD=dgcg-p4ssw0rd
      - ISSUANCE_COUNTRYCODE=CL
      - ISSUANCE_EXPIRATION_VACCINATION=365
      - ISSUANCE_EXPIRATION_RECOVERY=365
      - ISSUANCE_EXPIRATION_TEST=60
      # SERVICIOS DISPONIBLES
      - ISSUANCE_ENDPOINTS_FRONTENDISSUING=true
      - ISSUANCE_ENDPOINTS_BACKENDISSUING=true
```

```
      - ISSUANCE_ENDPOINTS_TESTTOOLS=true
      - ISSUANCE_ENDPOINTS_WALLET=true
      - ISSUANCE_ENDPOINTS_PUBLISHCERT=true
      - ISSUANCE_ENDPOINTS_DID=true
      # CONFIGURACION DE GATEWAY
      - DGC_GATEWAY_CONNECTOR_ENABLED=true
      - DGC_GATEWAY_CONNECTOR_ENDPOINT=https://lacpass.example.com:3050
      - DGC_GATEWAY_CONNECTOR_PROXY_ENABLED=false
      - DGC_GATEWAY_CONNECTOR_PROXY_HOST=
      - DGC_GATEWAY_CONNECTOR_PROXY_PORT=-1
      - DGC_GATEWAY_CONNECTOR_MAX-CACHE-AGE=300
      -
DGC_GATEWAY_CONNECTOR_TLSTRUSTSTORE_PATH=file:/app/certs/tls_trust_store
.p12
      - DGC_GATEWAY_CONNECTOR_TLSTRUSTSTORE_PASSWORD=dgcg-p4ssw0rd
      -
DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PATH=file:/app/certs/tls_key_store.p12
      - DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PASSWORD=dgcg-p4ssw0rd
      - DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_ALIAS=tls_key
      - DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PATH=file:/app/certs/ta.jks
      - DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PASSWORD=dgcg-p4ssw0rd
      - DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_ALIAS=trustanchor
      -
DGC_GATEWAY_CONNECTOR_UPLOADKEYSTORE_PATH=file:/app/certs/CL/upload_key_
store.p12
      - DGC_GATEWAY_CONNECTOR_UPLOADKEYSTORE_ALIAS=upload_key
      - DGC_GATEWAY_CONNECTOR_UPLOADKEYSTORE_PASSWORD=dgcg-p4ssw0rd
```

Make sure you replace the keys correctly. More information about this configuration in this link.

Run

To build the project executable, which is built through Maven, the following command is used:

```
$ mvn clean package
```

This will create a "jar" file in the "~/dgc-issuance-service/target" directory. To run the application you use:

```
$ java -jar target/dgc-issuance-service-latest.jar
```

And if you use Docker, you can use:

```
$ docker-compose up --build
```

At the end, the web service that issues certificates should be started on the port indicated in the configuration. For example, if port 8081 is used, you can navigate to this URL: http://localhost:8081/swagger

Web Client

To test the issuance of certificates, the DGC provides another repository called issuance-web. This is a web application that consumes the API delivered by the issuance-service and allows the generation of vaccination certificates. This application works independently if Testing mode or productive mode was chosen in issuance-service. To clone the repository, you run the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-issuance-web
```

Then to connect with the APIs it is necessary to modify the docker-compose.yml file, or change the nginx configuration file.

```
    - DGCA_ISSUANCE_SERVICE_URL=http://dgc-issuance-service:8081
    - DGCA_BUSINESSRULE_SERVICE_URL=http://dgc-businessrule-service:8082
```

Here you must specify the URLs of the issuance-service and business rule. Something important to note is that this repository brings these two services as dependencies, since in this guide we are setting up and configuring each service separately, it is necessary to remove these from the configuration.

Finally you can run the web application using the following command

```
$ docker-compose up --build
```

Verifier

This repository contains a backend that allows the verification of vaccination certificates issued. The repository can be cloned using the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-verifier-service
```

Keys

Like the issuance service, this repository needs the previously created keys.

Database

This repository uses a Postgresql database, if docker is not used to build the project, you need to install and create a Postgresql database.

Configurations

Like the issuance-service, the repository configuration is in the docker-compose.yml file, but the "src/main/resources/application.yml" file can also be changed directly in case of not using docker.

There is no testing mode for the verifier-service, so it is always used with an associated gateway. To configure it, it is necessary to modify the configuration file and indicate the routes to the gateway and the keys:

```
      - DGC_GATEWAY_CONNECTOR_ENDPOINT=https://dgc-gateway.example.com
      -
DGC_GATEWAY_CONNECTOR_TLSTRUSTSTORE_PATH=file:/ec/prod/app/san/dgc/tls_trus
t_store.p12
      - DGC_GATEWAY_CONNECTOR_TLSTRUSTSTORE_PASSWORD=dgcg-p4ssw0rd
      - DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_ALIAS=1
      -
DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PATH=file:/ec/prod/app/san/dgc/tls_key_st
ore.p12
      - DGC_GATEWAY_CONNECTOR_TLSKEYSTORE_PASSWORD=dgcg-p4ssw0rd
      - DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_ALIAS=ta
      -
DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PATH=file:/ec/prod/app/san/dgc/trust_anch
or.jks
      - DGC_GATEWAY_CONNECTOR_TRUSTANCHOR_PASSWORD=dgcg-p4ssw0rd
```

Run

Like the previous repositories, to build the project executable, the following Maven command is used:

```
$ mvn clean install
```

This will create a "jar" file in the "~/dgc-verifier-service/target" directory. To run the application you use:

```
$ java -jar target/dgc-issuance-verifier-latest.jar
```

And if you use Docker, you can use:

```
$ docker-compose up --build
```

At the end, the web service that issues certificates should be started on the port indicated in the configuration. For example, if port 8082 is used, you can navigate to this URL: http://localhost:8082/swagger

Verification Mobile Apps

For mobile applications the repositories are divided into the iOS and Android platforms. Both platforms have 4 repositories divided by functionalities that each one fulfills. For the development of applications to verify the certificates, only the “verifier” and “wallet” repositories will be modified according to what is needed. The repositories for both platforms are as follows:

- **App Core:** This repository contains all the services necessary to connect to the DGC Verifier Service and to the DGC Business Rule. It is also responsible for signing the certificates to be able to send them safely.
    - iOS: https://github.com/eu-digital-green-certificates/dgca-app-core-ios
    - Android: https://github.com/eu-digital-green-certificates/dgca-app-core-android

- **Verifier:** This repository contains the mobile application that is in charge of scanning and verifying the certificates using the public keys, it uses the App Core to make the pertinent calls.
    - iOS: https://github.com/eu-digital-green-certificates/dgca-verifier-app-ios
    - Android: https://github.com/eu-digital-green-certificates/dgca-verifier-app-android

- **Wallet:** This repository provides a user interface to manage and save personal DGCs.
    - iOS: https://github.com/eu-digital-green-certificates/dgca-wallet-app-ios
    - Android: https://github.com/eu-digital-green-certificates/dgca-wallet-app-android

- **CertLogic:** This repository contains the source code to handle CertLogic semantics in mobile applications.
    - iOS: https://github.com/eu-digital-green-certificates/dgc-certlogic-ios
    - Android: https://github.com/eu-digital-green-certificates/dgc-certlogic-android

In case QR code examples are required, there are these official examples (https://dgc.a-sit.at/ehn/testsuite).

IOS

The requirements for services on iOS are:

- A Mac or virtual machine is required to run Xcode.

- Xcode 12.5+ is used for builds. A macOS 11.0+ operating system is required.
- To install it on physical devices, an Apple developer account is required. For this you must enroll in the apple development program (Apple Developer Program)

Verifier

This repository contains the mobile application to verify certificates through iOS. In order to install this project you must first clone it locally with the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-verifier-app-ios
```

In order to have the connection and certificate signing services, you must also have the core repository in the same folder, you can use the same command used to clone the core repository.

```
<project folder>
|___dgca-app-core-ios
|___dgca-verifier-app-ios
```

Once you have both repositories installed, they must modify the context.jsonc file with the correct national system values. This file is located in the “context” folder. You must fill in the appropriate values as shown in the following diagram:

```
{
  // Origin in ISO alpha 2 code:
  "origin": "XX",
  "versions": {
    "default": {
      "privacyUrl": "https://<PRIVACY_URL>",
      "context": {
        "url": "https://<URL_ISSUANCE_SERVICE>/context",
        "pubKeys": [<PUBLIC_KEYS>]
      },
      "endpoints": {
        "claim": {
          "url": "https://<URL_ISSUANCE_SERVICE>/dgci/wallet/claim",
          "pubKeys": [<PUBLIC_KEYS>]
        },
        "countryList": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/countrylist",
          "pubKeys": [<PUBLIC_KEYS>]
        },
        "rules": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/rules",
          "pubKeys": [<PUBLIC_KEYS>]
```

```
        },
        "valuesets": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/valuesets",
          "pubKeys": [<PUBLIC_KEYS>]

        }
      }
    },
  }
}
```

Once you have these values, you can run the certificate validation application.

To modify the Locale of the app you just have to generate a new locale file inside the folder “Localization/DGCAVerifier”. Copy the file en.xloc and modify it to meet your localization.

Wallet

This repository contains the mobile application to save and manage personal certificates. In order to install this project you must first clone it locally with the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-wallet-app-ios
```

In order to have the connection and certificate signing services, you must also have the core repository in the same folder, you can use the same command used to clone the core repository.

```
<project folder>
|___dgca-app-core-ios
|___dgca-wallet-app-ios
```

Like the verifier, you must modify the context.jsonc in order to generate the personal certificate. You can also modify the location in the same file.

Android

The requirements for services on Android are:

- For development it is recommended to use Android Studio. The latest version available can be downloaded here.
- Android SDK version 26+

Verifier and Wallet (Android)

This repository contains the mobile application to verify certificates through Android. In order to install this project you must first clone it locally with the following command:

```
$ git clone
https://github.com/eu-digital-green-certificates/dgca-verifier-app-android
```

In order to have the connection and certificate signing services, you must also have the core repository in the same folder, you can use the same command used to clone the core repository.

```
<project folder>
|___dgca-verifier-app-android
|___dgca-app-core-android
|___dgc-certlogic-android
```

Once you have the repositories installed, they must modify the verifier-context.jsonc file with the correct national system values. This file is located in the “app/src/acc/assets” folder. You must generate a file called "config.json" in the same folder and fill in the appropriate values as shown in the following diagram:

Verifier

```
{
  // Origin in ISO alpha 2 code:
  "origin": "XX",
  "versions": {
    "default": {
      "privacyUrl": "https://<PRIVACY_URL>",
      "context": {
        "url": "https://<URL_VERIFIER_SERVICE>/context",
        "pubKeys": [<PUBLIC_KEYS>]
      },
      "endpoints": {
        "status": {
          "url": "https://<URL_VERIFIER_SERVICE>/signercertificateStatus",
          "pubKeys": [<PUBLIC_KEYS>]
        },
        "update": {
          "url": "https://<URL_VERIFIER_SERVICE>/signercertificateUpdate",
          "pubKeys": [<PUBLIC_KEYS>]

        },
        "countryList": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/countrylist",
          "pubKeys": [<PUBLIC_KEYS>]
```

```
        },
        "rules": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/rules",
          "pubKeys": [<PUBLIC_KEYS>]

        },
        "valuesets": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/valuesets",
          "pubKeys": [<PUBLIC_KEYS>]

        }
      }
    },
  }
}
```

Wallet

```
{
  // Origin in ISO alpha 2 code:
  "origin": "XX",
  "versions": {
    "default": {
      "privacyUrl": "https://<PRIVACY_URL>",
      "context": {
        "url": "https://<URL_ISSUANCE_SERVICE>/context",
        "pubKeys": [<PUBLIC_KEYS>]
      },
      "endpoints": {
        "claim": {
          "url": "https://<URL_ISSUANCE_SERVICE>/dgci/wallet/claim",
          "pubKeys": [<PUBLIC_KEYS>]
        },
        "countryList": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/countrylist",
          "pubKeys": [<PUBLIC_KEYS>]
        },
        "rules": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/rules",
          "pubKeys": [<PUBLIC_KEYS>]

        },
        "valuesets": {
          "url": "https://<URL_BUSINESSRULE_SERVICE>/valuesets",
          "pubKeys": [<PUBLIC_KEYS>]

        }
      }
    },
  }
}
```

In the file "app/src/main/java/dgca/verifier/app/android/di/NetworkModule.kt" modify the variable "BASE_URL" by the url of the verifier:

```
const val BASE_URL = "https://<URL_VERIFIER_SERVICE>/"
```

In the case of the wallet, change it to:

```
const val BASE_URL = "https://<URL_ISSUANCE_SERVICE>/"
```

To run the project in an android emulator you must execute this command:

```
$ gradlew -PCONFIG_FILE_NAME="config.json"
```

Frequent questions

This section describes the most frequently asked questions related to the project

- If I am using the issuance-web, how can I add authentication or some access restriction measure?

The issuance web application (issuance-web) does not have its own authentication system or any access control measure. This means that once this application is deployed, any user with access to the server can use it and issue certificates. There are different strategies to control access depending on the complexity and amount of resources that the countries have.

The simplest alternative to implement is to add basic HTTP authentication to the proxy server either nginx or apache. Both provide plugins to add this type of authentication, for example by configuring the .htpasswd file.

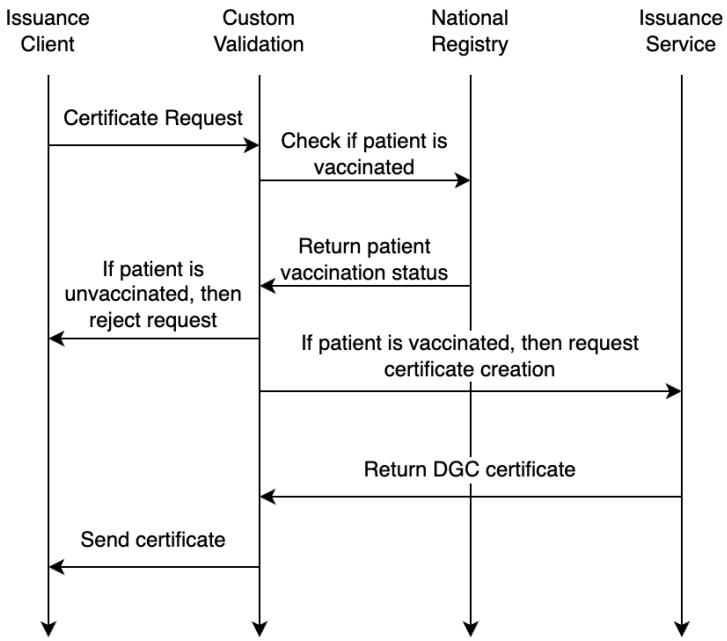
The next recommended option is not to use the issuance-web service and develop your own application that uses the issuance-service API. This way you have full control of development and you can add authentication at the application level.

In any case, it is recommended that the application is only visible for a specific segment of IPs or locations to avoid other types of traffic.

- If I have a national registry of vaccinated people, how can I integrate this information into the system?

Countries with a National Vaccination Registry can connect the information of vaccinated people to LACPASS through the issuance-service API. It is recommended to create software that is

inserted between the certificate issuance client (issuance-web or custom development) and the issuance-service. This software should verify the authenticity of the certificate requested to be issued with the national registry and continue with the issuance process or reject it in case the requested data does not match. The following diagram exemplifies the operation.



- What are the steps to follow to be able to integrate with the EU?

The integration process is explained in the following link: Onboarding Checklist.

In general, the process consists of sending the public keys to the EU to add them to the gateway database as explained in the gateway section. And after testing that everything is working correctly, you need to change the gateway endpoint to their official endpoint.

Ellos tienen a disposición 3 ambientes: Test, para las pruebas de integración (este ambiente sólo se inicia cuando se empieza el proceso de Onboarding, antes de eso está apagado). Acceptance, para probar y para que la UE valide que la integración funciona correctamente. Producción: ambiente que contiene los datos reales, una vez integrado a este ambiente se completa el proceso.

- How to handle people who are vaccinated for the first time and people who are already vaccinated?

It is recommended that countries that have implemented a National Vaccination Registry have integrated it using the instructions above. In this way, the custom validation layer that will be

developed can handle cases in which people have already been vaccinated, have been vaccinated for the first time or have an incomplete vaccination schedule.