Río Negro
Universidad Nacional

Trabajo Final de Carrera

Despliegue de un entorno de credenciales verificables (VC) bajo el estándar de la W3C

Autor: Carlos Leandro Isaias Farra Gomez

Director: Mauro Cambarieri

Codirector: Guillermo Malpeli

Índice

Trabajo Final de Carrera **1**
Índice **2**
1. Breve marco teórico **3**
1.1 Claim 3
1.2 Credencial Verificable (VC) 3
1.3 JSON-LD 4
1.4 DID 4
1.5 Wallet 5
1.6 Consorcio 5
1.7 Tenant 5
2. Tareas realizadas **5**
2.1 Análisis 5
2.2 Diseño de la solución 7
2.3 Implementación 11
2.4 Documentación 12
3. Procesos del entorno de Credenciales Verificables **13**
3.1 Emisión de un lote de Credenciales Verificables 13
3.2 Almacenamiento en la Wallet de una Credencial Verificable 14
3.3 Verificación de una Credencial Verificable 15
4.Resultados **16**
4.1 Ecosistema resultante 16
5. Fuentes **17**
5.1 Bibliografía 17
5.2 Software utilizado 17
6. Anexos **17**
6.1 Archivos de despliegue en local 17
6.2 Archivos de ejemplo 23
6.3 Ejemplos de código 28

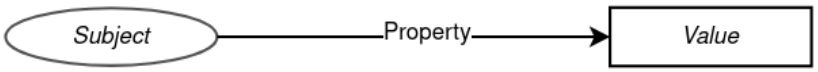
En este trabajo final de carrera, se busca detallar el despliegue de un entorno de Credenciales Verificables (VC) bajo el estándar Verificable Credentials Data Model v2.0 (World Wide Web Consortium [W3C], 2025).

1. Breve marco teórico.

Se incluyen en este marco teórico las definiciones de los conceptos principales del dominio de trabajo.

1.1 Claim

Una claim es la unidad básica de información dentro del modelo de datos. Representa una declaración sobre un sujeto, el cual puede ser una persona, una organización u otro tipo de entidad. Estas afirmaciones se estructuran mediante relaciones del tipo “sujeto - propiedad - valor”, lo que permite expresar una amplia gama de hechos, como por ejemplo, si una persona ha completado estudios en una universidad específica.

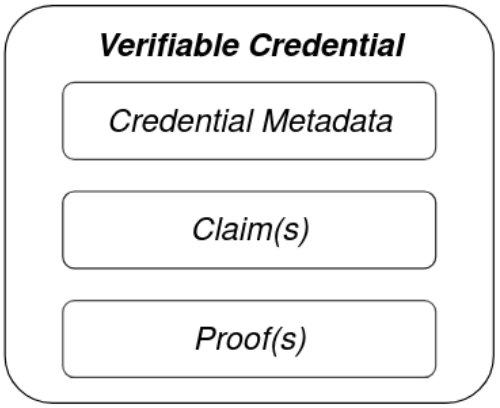


Fuente: Elaboración personal a partir de la W3C.

1.2 Credencial Verificable (VC)

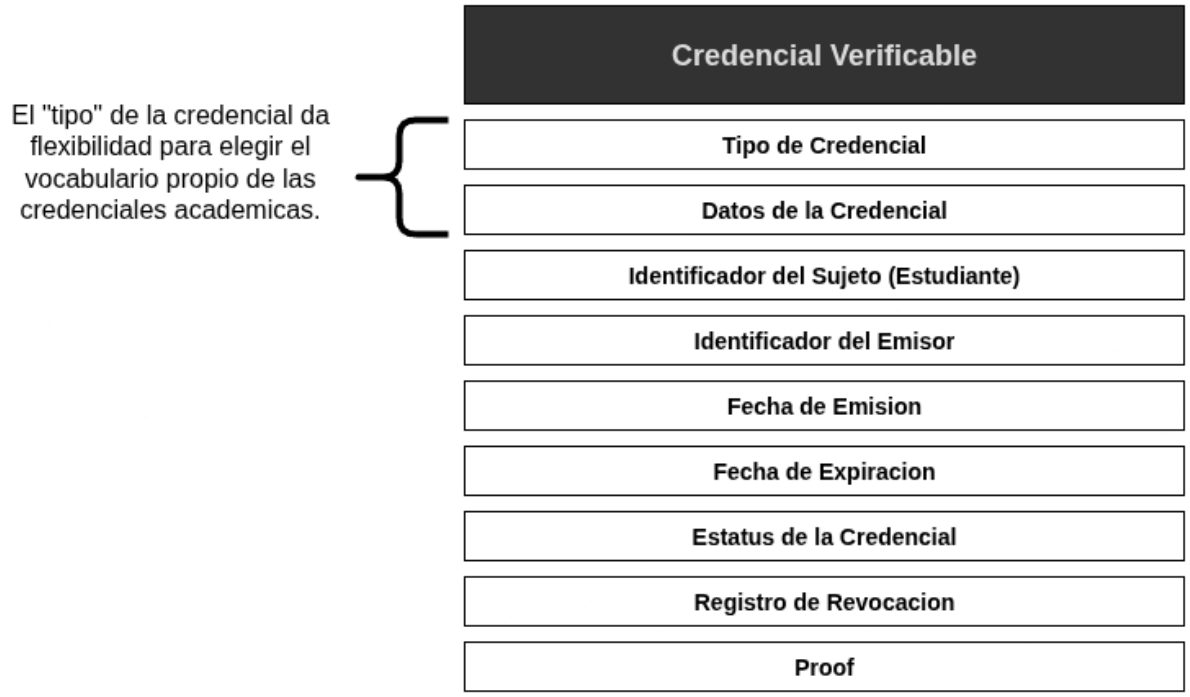
Una credencial verificable es una credencial enriquecida con mecanismos criptográficos que permiten probar su origen y detectar cualquier alteración. De esta forma, el modelo garantiza tanto la autenticidad del emisor como la integridad del contenido. Ejemplos típicos de credenciales verificables incluyen identificaciones digitales de empleados, licencias de conducir digitales y certificados académicos digitales.

Está compuesta por una agrupación de una o más *claims* que provienen de una misma entidad emisora; un proof; y diversos metadatos para definir su estructura.



Fuente: Elaboración personal a partir de la W3C.

Entre los metadatos mas comunes que suele contener una credencial verificable están:



Fuente: Elaboración personal a partir de la W3C.

1.3 JSON-LD

La especificación de Credenciales Verificables utiliza *JSON-LD 1.1* como formato de serialización para representar su modelo de datos basado en grafos. JSON-LD facilita la representación semántica y extensible de estos datos en un formato familiar para desarrolladores que ya usan JSON.

Tiene características como permitir que las credenciales sean representadas como un documento JSON; palabras clave como *@id* o *@type* para definir de forma más semántica los campos clave; soporte para tipos extra como fechas, url's, entre otros.

1.3.1 JSON

JavaScript Object Notation es un lenguaje para definir objetos de forma concisa y simple. Soporta campos con valores simples (números enteros; números flotantes; cadenas de texto; booleanos), valores compuestos (otro objeto JSON anidado) y colecciones simples (arrays y diccionarios).

1.4 DID

El *Descentralizaded ID* es el identificador único con el que una institución puede firmar una credencial verificable. De esta forma garantiza que las credenciales verificables se pueden validar sin depender únicamente de la entidad que la emitió. Consiste de 2 elementos:

- Una **seed**, que cada entidad debe almacenar de forma confidencial, ya que se usa en el servicio de emisión para firmar las credenciales verificables.
- El **did** propiamente dicho, que se guarda en un registro común entre todas las entidades que conformen el consorcio establecido. En nuestro caso, al ser un

prototipo, representamos la única entidad registrada. Este sirve para ser consultado por las instancias del verificador y validar que una determinada credencial verificable fue emitida por una entidad perteneciente al registro.

1.5 Wallet

La wallet es un espacio de almacenamiento donde el usuario puede guardar sus credenciales. La puede usar para mostrar la credencial, o una presentación parcial. En la implementación del entorno está representada por una aplicación de Android.

1.6 Consorcio

Se usará este término para abarcar a las entidades que conforman el registro de DIDs, representa un conjunto de entidades capaces de emitir credenciales verificables y validar si los demás miembros del consorcio emiten sus propias credenciales.

1.7 Tenant

Es una entidad autorizada para emitir credenciales verificables en el entorno del consorcio. No es precisamente un concepto del dominio sino que forma parte del proyecto.

2. Tareas realizadas

2.1 Análisis

2.1.1 Situación actual

Las universidades, o cualquier institución educativa, convalida los conocimientos de los estudiantes mediante la emisión de certificados o títulos. Actualmente, esta convalidación se hace al completar un curso o una carrera. Esto quiere decir que en el sistema actual no existe la posibilidad de acreditar la adquisición de conocimientos parciales sino solamente finalizados.

En el caso de los cursos, las certificaciones suelen consistir en un archivo *pdf*, fácil de perder y de presentar. Se han tomado medidas para mitigar estos déficits., Una de ellas es la verificación a través de un código QR que se contrasta contra un servicio de verificación propio de la entidad. Sin embargo, esto conlleva una desventaja: se necesita un verificador centralizado que depende única y exclusivamente de que el emisor lo mantenga funcionando. Si eventualmente se da de baja, no se podrían verificar más los certificados emitidos por dicha entidad.

En el caso de las carreras de grado, la certificación a la finalización del cursado representa la aprobación de la totalidad de la malla curricular. Esto genera que la oferta académica resulte bastante inflexible y monolítica, ya que el estudiante no puede validar su conocimiento en determinadas áreas, hasta completar el título de grado.

2.1.2 Infraestructura disponible

Para el desarrollo del proyecto, se hace uso de los recursos del Laboratorio de Informática Aplicada (LIA) de la UNRN. Dicho laboratorio cuenta con los siguientes componentes en su infraestructura:

- Nodo local de GitLab, como sistema de control de versiones. El mismo cuenta con su propio *runner* configurado, de forma que la pipeline se ejecuta en el mismo servidor.
- Nodo local de Docker Swarm, para el despliegue de las imágenes OCI generadas con el runner de GitLab, que permiten definir y configurar la infraestructura de los servicios de forma más sencilla.
- Servicio SMTP, que nos proporciona un email con el que enviar las credenciales de prueba.
- Servicio de DNS propio, para generar dominios con los que exponer y probar los servicios en el internet público.

2.1.3 Herramientas de Software utilizadas

Para el desarrollo del proyecto, es necesario o recomendable el uso de diversas aplicaciones de software. Estas se encargan de aspectos como el tratamiento del código de las aplicaciones a desplegar, el uso de los componentes de la infraestructura.

Entre las herramientas para trabajar con el código están:

- **Visual Studio Code**, un editor de texto que soporta varios lenguajes de programación.
- **Meld**, una herramienta que permite comparar los archivos de 2 directorios. De forma que permite encontrar fácilmente las diferencias entre 2 versiones distintas de la misma aplicación, y elegir con cual quedarte. Esta diferenciación la puede hacer a nivel de sistema de archivos, de línea dentro de un archivo, y de carácter dentro del archivo. Permitiendo comparar con el nivel de granularidad que resulte más conveniente para cada caso.

Para el uso adecuado y cómodo de la infraestructura, se usó:

- **Git**, sistema de control de versiones.
- **Portainer**, un servicio web que permite gestionar con interfaz gráfica el servidor de Docker Swarm.

2.1.4 Limitaciones y alcance

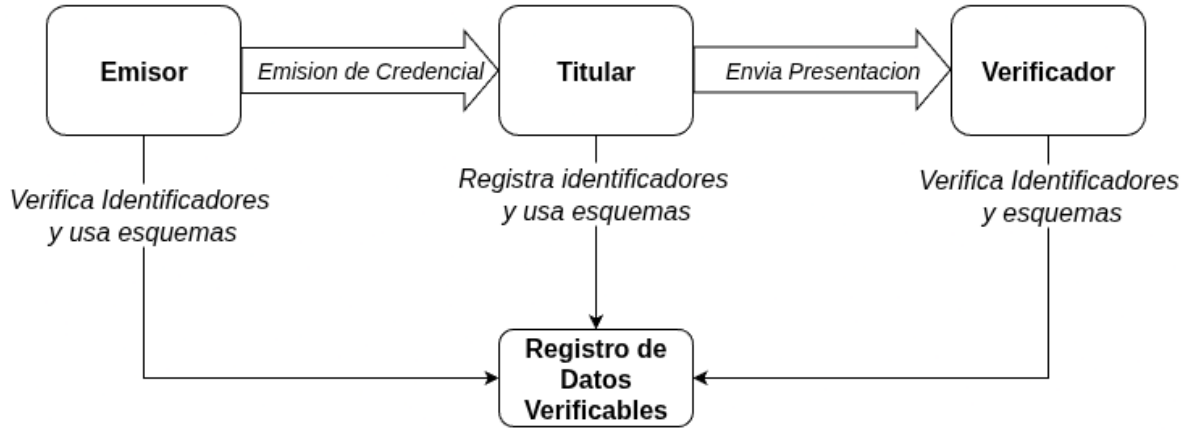
Al tratarse de un prototipo, se desestima la implementación de funcionalidades que mejoran la experiencia de usuario, tales como autenticación con OAuth 2, recuperación de cuenta. Se prioriza la funcionalidad principal, que es la emisión, almacenamiento y verificación de las credenciales.

2.2 Diseño de la solución

2.2.1 Credenciales Verificables

Ante estas limitaciones, surgen como alternativa las credenciales verificables que ofrecen almacenar esa información de forma confiable, portable y verificable. De forma descentralizada e interoperable.

La W3C establece el estándar *Verifiable Credentials Data Model v2.0* con los siguientes actores:



Fuente: Elaboración personal a partir de la W3C.

2.2.1.1 Emisor

Es la entidad responsable de generar credenciales a partir de afirmaciones sobre un sujeto determinado. Este rol puede ser desempeñado por organizaciones, instituciones públicas o individuos que emiten documentos digitales verificables.

2.2.1.2 Titular

Es quien recibe, almacena y presenta sus credenciales ante terceros. En muchos casos, el titular coincide con el sujeto de la credencial (por ejemplo, una persona que recibe un diploma), aunque no es una condición obligatoria.

2.2.1.3 Verificador

Representa al actor que recibe y procesa credenciales con el fin de verificar su validez. Para ello, se apoya en mecanismos criptográficos, registros confiables y, eventualmente, presentaciones verificables. El verificador toma decisiones basadas en la autenticidad y vigencia de las credenciales recibidas, sin depender necesariamente de una relación preexistente con el emisor. Algunos ejemplos comunes incluyen empleadores, portales web, o entidades de control de acceso.

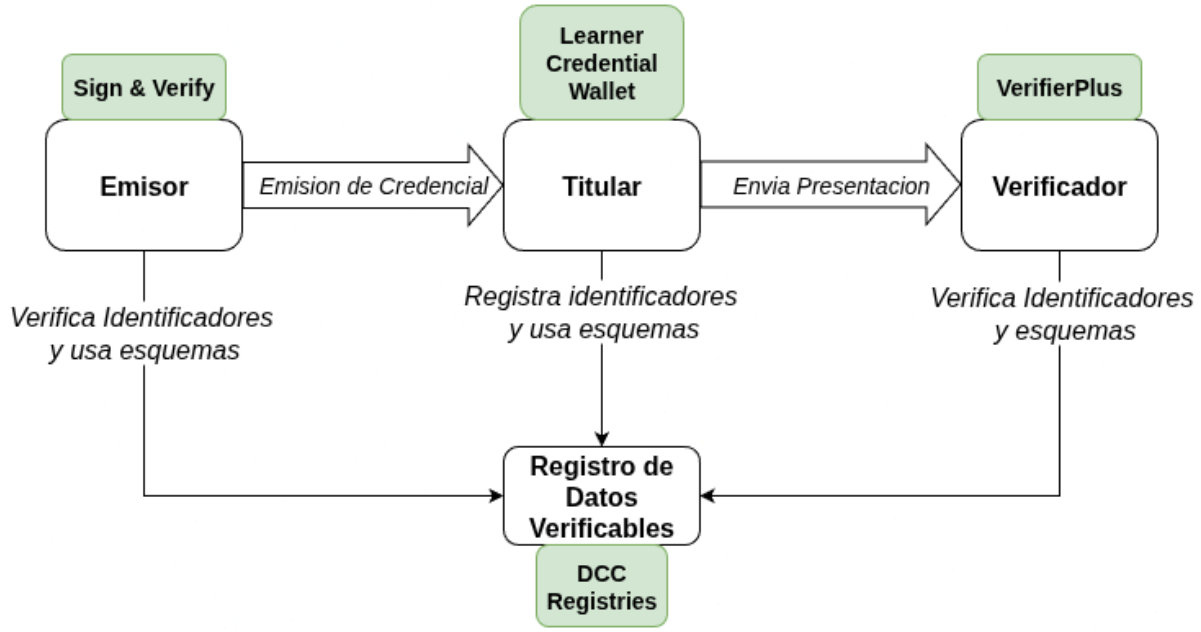
2.2.1.4 Registro de Datos

Son infraestructuras utilizadas para facilitar la verificación de las credenciales. Pueden incluir esquemas de datos, claves públicas, listas de revocación, y otros mecanismos de

soporte. Estos registros pueden ser centralizados, federados o distribuidos (por ejemplo, blockchain, IPFS).

2.2.2 Soluciones preexistentes

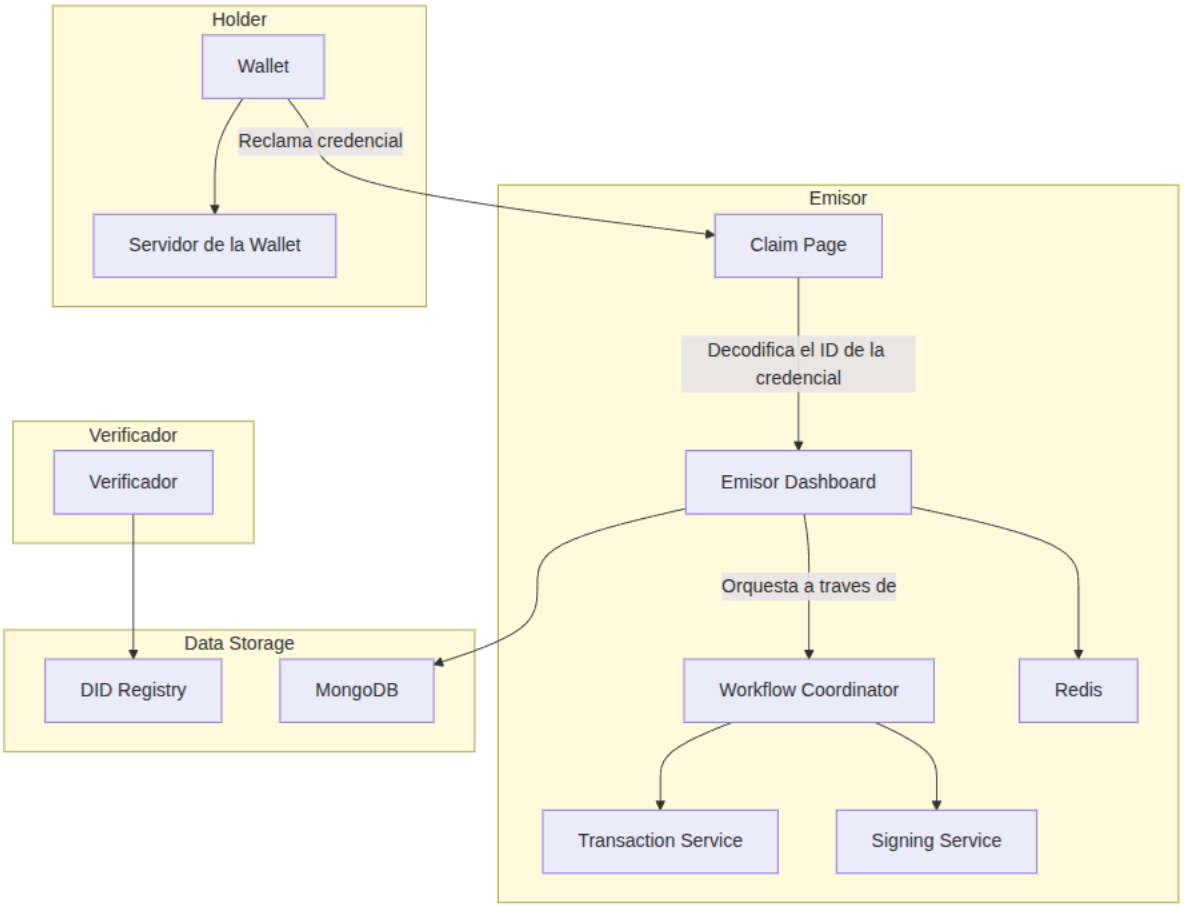
Para un desarrollo más ágil, sin descuidar la robustez, se optó por tomar un ecosistema ya existente con licencia Open Source e implementarlo. El mismo fue desarrollado por el *Digital Credential Consortium (DCC)*.
La solución ofrece un conjunto de servicios, desarrollados con JavaScript y sus diversos frameworks, como: Express, Next, Expo, React. De forma que se implementan los actores necesarios, propuestos por la W3C: Emisor, Dueño, Verificador y Registro de Datos.



Fuente: Elaboración personal a partir de la DCC.

2.2.3 Arquitectura del ecosistema

Cada uno de estos actores representa un proyecto de software. La wallet (titular) y el VerifierPlus (verificador) son aplicaciones monolíticas, es decir, que su lógica de negocio está contenida en un solo módulo. Mientras que el servicio de emisión consiste de una aplicación con arquitectura de microservicios, lo cual significa que son varios módulos interdependientes.



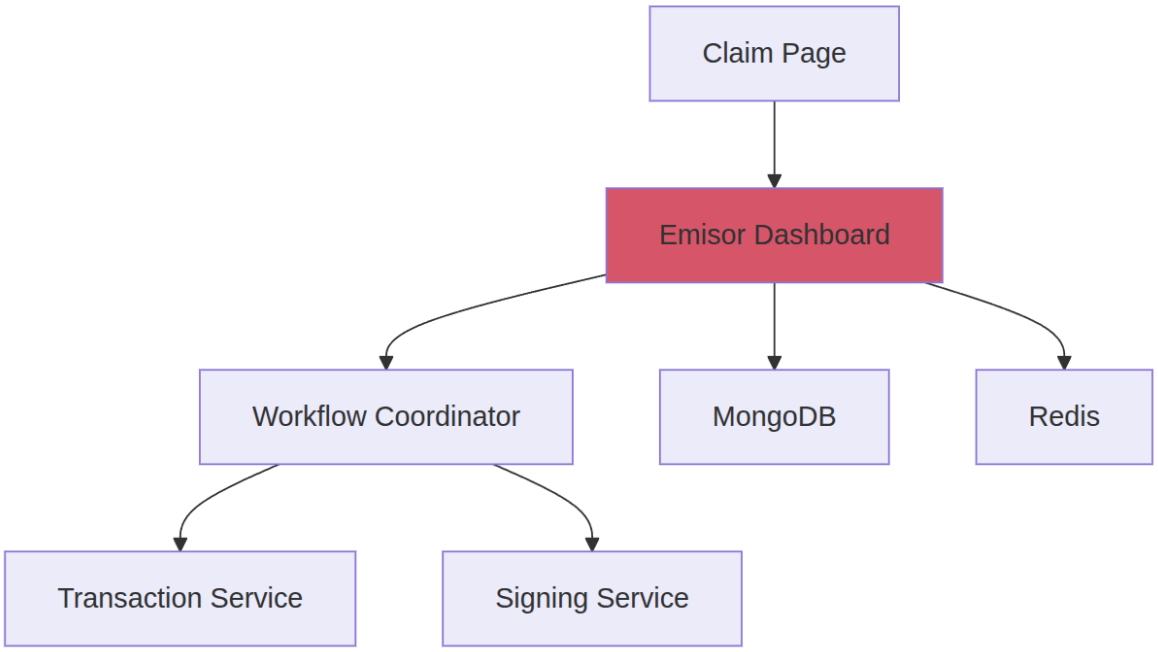
Fuente: Elaboración personal

2.2.4 Arquitectura del proyecto Sign and Verify (Emisor)

La arquitectura del proyecto consta de cinco servicios:

1. Claim page
2. Dashboard
3. Workflow coordinator,
4. Transaction service,
5. Signing service.

También cuenta con un repositorio de datos que utiliza tecnología Mongo DB y un servicio de caché utilizando Redis...



Fuente: Elaboración personal

2.2.4.1 Dashboard

Un dashboard es donde el administrador del sistema puede gestionar varios aspectos relacionados con la emisión de credenciales verificables, trabajando por lotes. Estas funcionalidades incluyen:

- Gestión de plantillas (templates) para las credenciales verificables. En estas, se pueden modificar los campos para incluir los que se crean convenientes.
- Gestión de plantillas para los emails que se envían a los alumnos para reclamar su credencial verificable. Se escribe como formato *Mustache*, una sintaxis basada en HTML para definir templates con variables que pueden reemplazarse posteriormente.
- Emisión de lote de credenciales, el administrador elige una de las plantillas de credenciales definida previamente, luego carga un archivo .csv que satisfaga las columnas de datos de la plantilla, una plantilla de email, se confirma la operación y la creación del lote se ejecuta.
- Gestión de historial de lotes que permite ver las credenciales emitidas, a quienes, en qué estado está el lote mismo, reenviar los mails.

2.2.4.2 Claim Page

Este sitio es donde el usuario que recibe una credencial verificable a través de su correo electrónico, puede reclamar para almacenarla en su wallet personal.

2.2.4.3 Workflow Coordinator

Coordina el flujo de emisión de los lotes de credenciales verificables, orquestando el *Signing Service* y el *Transaction Service*.

2.2.4.4 Signing Service

Se encarga de firmar las credenciales verificables, usando la *seed* correspondiente a la entidad que solicite la emisión.

2.2.4.5 Transaction Service

Se encarga de almacenar cada lote de credenciales emitidas y firmadas, garantizando la transaccionalidad de la operación.

2.3 Implementación

2.3.1 Configuración de la Infraestructura.

2.3.1.1 Repositorio

Los proyectos fueron gestionados por el servicio de GitLab del LIA- UNRN para almacenar el código y las versiones de los mismos.

2.3.1.2 Integración Continua

Se configuró la pipeline para los proyectos de emisor y verificador, de forma que se construye una imagen OCI que puede ser desplegada tanto en el servidor de Docker de la UNRN como en una computadora personal para pruebas.
En el caso de la wallet, la pipeline genera un archivo APK, para poder integrarlo con la tienda.

2.3.1.3 Despliegue

Se desplegó el emisor y el verificador sobre el servidor de Docker de la UNRN, se les asignó un dominio para que sean visibles públicamente en internet. En el caso de la wallet, se generó un archivo APK para instalarlo en dispositivos android, y se requiere un servidor web activo donde la wallet del usuario debe poder conectarse.

2.3.2 Actualización de los servicios.

Las versiones alpha con las que trabajamos inicialmente, quedaron rápidamente atrasados respecto a los repositorios originales de DCC. Es necesario actualizar los repositorios locales, tanto por razones de seguridad (las dependencias viejas tienen vulnerabilidades conocidas) como de funcionalidad (corrección de errores, nuevas funcionalidades).
Pero también es importante mantener las customizaciones realizadas, que consisten en traducción (los repositorios originales tienen los textos en inglés), paleta de colores personalizada.

2.3.3 Creación de un registro de DIDs propio

Para poder conformar nuestro propio consorcio, que se incluyó en la Sección 1.6 es necesario consolidar nuestro propio registro de DIDs. Un registro de DIDs se define en un documento JSON que consta de un array de objetos, donde cada objeto representa a una entidad perteneciente al consorcio, y por tanto, con capacidad para emitir sus propias

credenciales verificables. El did funciona como una llave pública que valida que un conjunto de datos fue encriptado usando una llave privada, la seed.
El servicio de signing usa una de las *seeds* que tenga definida como variable de entorno para firmar las credenciales. Y también permite la creación de nuevos dids, el cual consta de la seed que se usará para firmar, y debe mantenerse confidencial, y el did que se expondrá en el registro de dids. La creación se puede realizar por 2 métodos:

- **did:key** no recibe parámetros de entrada, y devuelve un documento JSON con 2 atributos: seed y did.
- **did:web** recibe una url, y devuelve un documento JSON con 2 atributos: seed y did. Este método requiere que la url definida se mantenga activa y accesible en un host

Se usó el servicio de signing para crear un did con el método did:key. Este método no pide parámetros de entrada, y devuelve un JSON con 2 datos claves:

- **seed**: que se establece como variable de entorno en el propio servicio de signing. Usa este dato para firmar las credenciales de un determinado emisor.
- **did**: se debe guardar en el *registry.json* correspondiente al consorcio. Este es consultado por el verificador para validar que una credencial verificable fue emitida por un emisor confiable, perteneciente al consorcio.

2.4 Documentación

Dado la escala del proyecto, separado en varios servicios, se documentó en los repositorios de diversas maneras. Ya sea a través de documentos Markdown (.md), archivos de despliegue y configuración de ejemplo y también para desplegarlo en un entorno de desarrollo en máquina local.

2.4.1 Metodologías utilizadas para documentar el desarrollo

2.4.1.1 Documentos Markdown

A través de documentos con formato Markdown, se documentaron procesos, configuraciones, diagramas de arquitectura y conceptos a tener en cuenta. Tanto a nivel de servicio individual como el conjunto de cada uno de los actores. Los mismos cuentan con gráficos escritos con sintaxis Mermaid, la cual es soportada por el servidor de GitLab de la UNRN.

2.4.1.2 Archivos de ejemplo

Durante el desarrollo, se generaron archivos *docker-stack.example.yml* que se corresponden en configuración, pero no en valores, a los implementados en el servidor de docker de la UNRN. Se dejaron archivos *.example.env* que definen las variables de entorno necesarias. Ver anexo 4.1

2.4.1.3 Despliegue en entorno de pruebas

Se crearon archivos *docker-compose.yml* para posibilitar un despliegue en un entorno local. Ver anexo 4.2

2.1.1.4 Documentacion de DCC

La documentación original de los proyectos de DCC se movía a una carpeta *docs/legacy*.

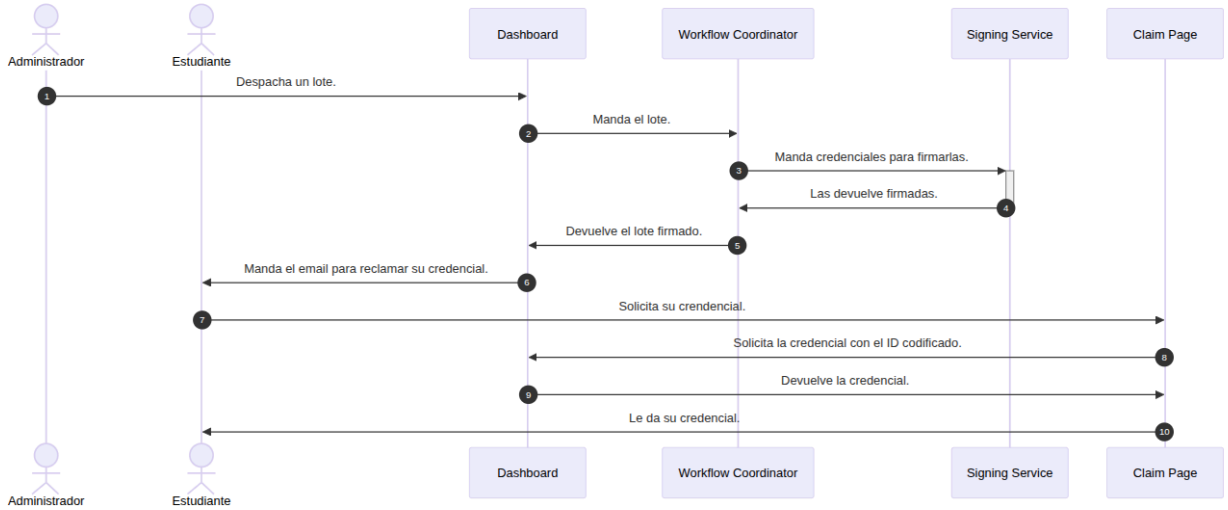
3. Procesos del entorno de Credenciales Verificables

Con el ecosistema de credenciales verificables desplegado en su totalidad, se pueden realizar los siguientes procesos para hacer uso del mismo.

3.1 Emisión de un lote de Credenciales Verificables

Las credenciales verificables se emiten como un lote donde se requiere un template de la credencial, un template del mail a enviar y un conjunto de datos que se corresponda con los templates anteriormente mencionado.

Diagrama de Secuencia de la emisión de una Credencial Verificable



Fuente: Elaboración personal

3.1.1 Plantillas requeridas

Se necesita tener definida una plantilla para la Credenciales Verificables y para el email de notificación. En el anexo adjunto un ejemplo de ambos.

3.1.2 Definición de los datos del lote

Se define un CSV que satisfaga los datos solicitados tanto en la plantilla de la credencial como en la plantilla del email.

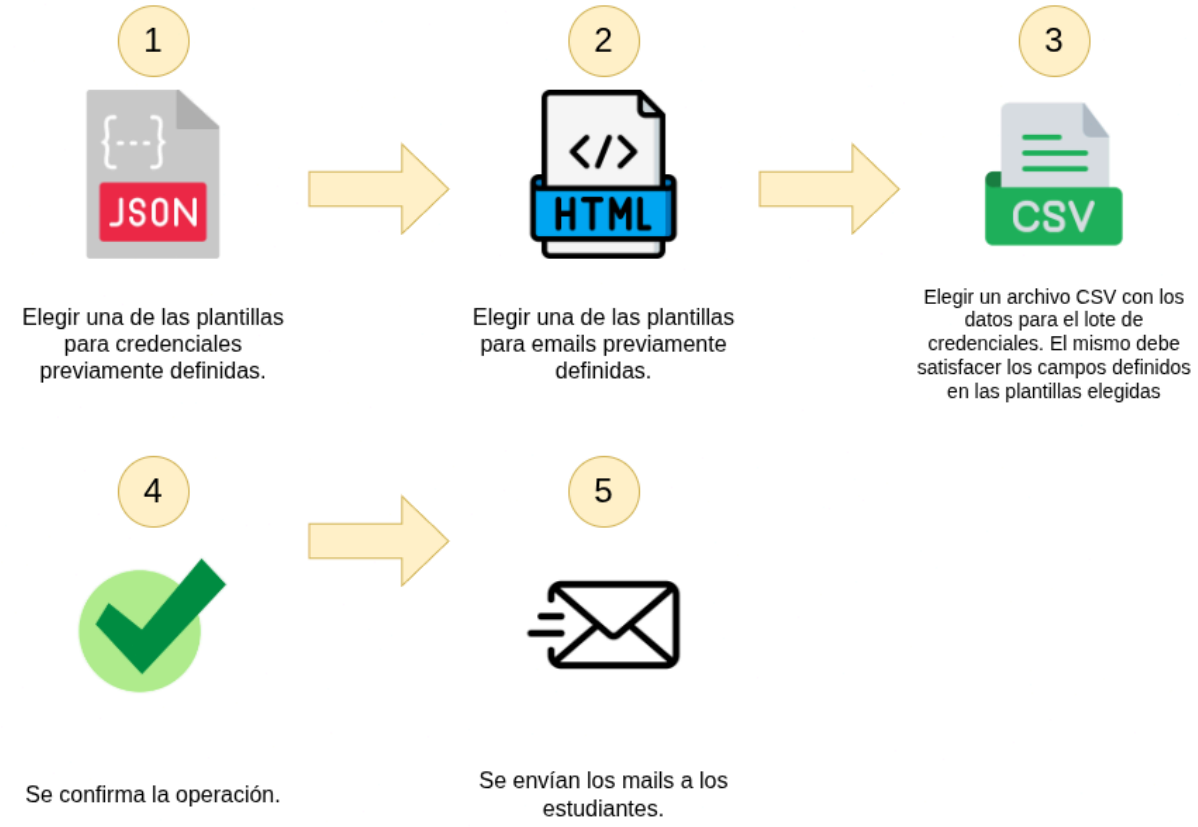
3.1.3 Confirmación de los datos

Se verifica que el conjunto de datos se corresponda correctamente con las plantillas seleccionadas, y si da todo bien, se puede emitir el lote.

3.1.4 Envío del lote

Una vez que todo esté en orden, se envía el lote de credenciales solicitado para ser emitido y se notifica a los alumnos por su email.

Pasos para emitir un lote de Credenciales Verificables



Fuente: Elaboración personal

3.2 Almacenamiento en la Wallet de una Credencial Verificable

Una vez que el estudiante recibe la notificación por mail, este puede reclamar su credencial verificable y almacenarla en su wallet personal.

3.2.1 Recepción del Email

El estudiante recibe un email, el mismo contiene un enlace al sitio de claim page, donde hay un QR con el que puede reclamar su credencial verificable.

3.2.2 Almacenamiento en la Wallet

Con la aplicación de la wallet, se puede leer el QR para solicitar los datos de la credencial verificable, y así almacenarla.

Fuente: Elaboración personal

3.3 Verificación de una Credencial Verificable

3.3.1 Abrir el sitio del VerifierPlus

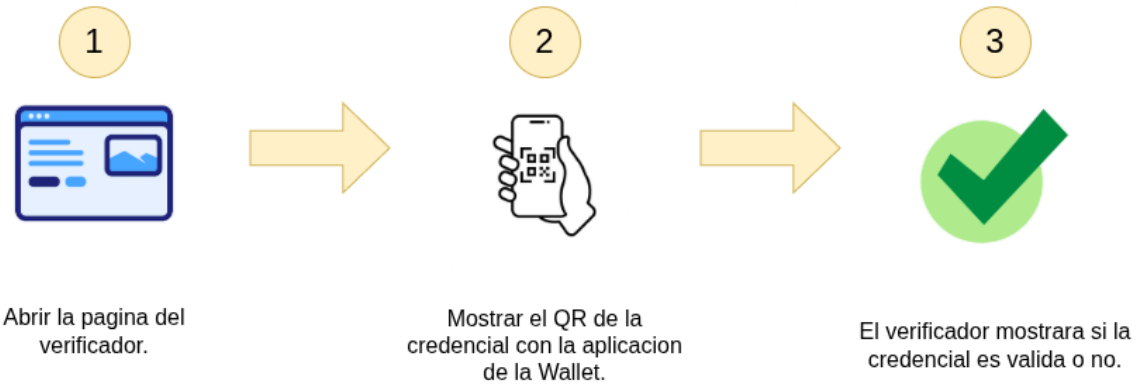
Se va al sitio del VerifierPlus de alguna entidad perteneciente al mismo consorcio que quien la emitio.

3.3.2 Exponer la credencial al verificador

Con la app de la Wallet se puede mostrar la credencial en formato de QR, o también extraer el JSON con que la representa. El proyecto VerifierPlus permite ambas formas para validar la credencial.

El verificador devolverá si la credencial es válida o no.

Pasos para verificar una Credencial Verificable



Fuente: Elaboración personal

4.Resultados

4.1 Ecosistema resultante

El ecosistema de software conseguido cumple con las funcionalidades mínimas que se requieren para la gestión de credenciales verificables, que constan de: emisión, almacenamiento y verificación descentralizada.

4.1.1 Emisión

El servicio de emisión puede emitir credenciales verificables por lote, usando plantillas para las propias credenciales y para los mails de notificación a quienes les corresponde. Las firma usando la seed privada de la entidad, que se corresponde en el registro con un DID público.

4.1.2 Almacenamiento

El usuario final puede almacenar las credenciales verificables emitidas y compartirlas para ser verificadas por un verificador perteneciente al consorcio.

4.1.3 Verificación

El servicio de verificación puede validar si una credencial fue emitida por una entidad perteneciente al registro de DIDs público. Si bien el alcance del proyecto se limitaba a una sola entidad, el método de verificación a través de DIDs públicos permite escalarlo a más de una entidad de forma sencilla.

5. Fuentes

5.1 Bibliografía

W3C Verificables Credentials Data Model
https://bit.ly/4oVmotK
DCC - Digital Credentials Consortium
https://bit.ly/3LVCchl

5.2 Software utilizado

Meld, herramienta para realizar merge entre 2 directorios o archivos. GPLv2
https://bit.ly/3XIbI5C
Visual Studio Code, editor de código. MIT License.
https://bit.ly/47Ub62N

6. Anexos

6.1 Archivos de despliegue en local

En esta sección se describen los archivos docker-compose.yml que permiten desplegar cada servicio en máquina local de forma simple, usando Docker Compose.

6.1.1 Emisor

```
services:
  coordinator:
    image: digitalcredentials/workflow-coordinator:0.1.0
    environment:
      - ENABLE_STATUS_SERVICE=
      - PUBLIC_EXCHANGE_HOST=
      - TENANT_TOKEN_LEF_TEST=
    ports:
      - '4005:4005'
  signing:
    image: digitalcredentials/signing-service:0.2.0
    environment:
      - TENANT_SEED_{TENANT_NAME}=
  transactions:
    image: digitalcredentials/transaction-service:0.1.0
  payload:
    build:
      context: .
      dockerfile: Dockerfile.local
    depends_on:
      - coordinator
      - redis
      - mongo
    environment:
      - COORDINATOR_URL=
      - REDIS_URL=
      - REDIS_PORT=
      - MONGODB_URI=
      - PAYLOAD_SECRET=
      - TENANT_NAME=
      - SMTP_HOST=
      - SMTP_USER=
      - SMTP_PASS=
      - SMTP_STARTTLS=
      - EMAIL_FROM=
      - CLAIM_PAGE_URL=
      - PAYLOAD_PUBLIC_SERVER_URL=
```

```
    ports:
      - '3000:3000'
    volumes:
      - ./src:/home/node/app/src
  claim-page:
    image: digitalcredentials/admin-dashboard-claim-page:0.1.0
    container_name: 'ad-claim-page'
    depends_on:
      - payload
    ports:
      - '8080:8080'
  redis:
    image: redis:alpine
    container_name: ad-redis
    environment:
      - ALLOW_EMPTY_PASSWORD=
    ports:
      - '6379:6379'
  mongo:
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME:
      MONGO_INITDB_ROOT_PASSWORD:
    ports:
    - "27017:27017"

  mongo-express:
    image: mongo-express:latest
    container_name: mongo-express
    ports:
    - "8081:8081"
    environment:
      ME_CONFIG_MONGODB_SERVER: mongo
      ME_CONFIG_MONGODB_PORT:
      ME_CONFIG_MONGODB_ADMINUSERNAME:
      ME_CONFIG_MONGODB_ADMINPASSWORD:
    depends_on:
    - mongo
volumes:
  transactions:
```

Las variables de entorno de cada servicio sirven para configurar:

6.1.1.1 Variables de Workflow Coordinator

Nombre de la variable	Valor esperado	Para qué sirve

ENABLE_STATUS_SERVICE	false | true	Define si se levantó el status service.
PUBLIC_EXCHANGE_HOST	String de URL o dirección web	URL del dashboard del emisor
TENANT_TOKEN_LEF_TEST	Un string que representa un token | UNPROTECTED	Token que usará el emisor para validar su identidad. UNPROTECTED para que no lo haga.

6.1.1.2 Variables del Signing Service

Nombre de la variable	Valor esperado	Para qué sirve
TENANT_SEED_{TENANT_NAME}	Un string que representa la seed del tenant	Es lo que usa para firmar las credenciales verificables. El TENANT_NAME es el nombre de una entidad autorizada para firmar.

6.1.1.3 Variables del Dashboard

Nombre de la variable	Valor esperado	Para qué sirve
COORDINATOR_URL	URL	URL del Workflow Coordinator.
REDIS_URL	URL	URL de Redis, una base de datos clave-valor usada como caché.
MONGODB_URI	URL	URL de Mongo, donde se incluye el hostname, el puerto, el usuario y la contraseña para loguearse.
PAYLOAD_SECRET	String	Salt usado para codificar y decodificar el ID de una credencial verificable recién emitida y enviada por correo a su respectivo estudiante. Es usada por el claim page una vez que se reclama la credencial.
TENANT_NAME	String	Nombre de la entidad emisora de la credencial verificable. Se usa para

		que el Signing Service sepa que seed debe usar.
SMTP_HOST	String	Host del servidor SMTP que envía los correos.
SMTP_USER	String	Usuario del servidor SMTP que envía los correos.
SMTP_PASS	String	Contraseña del usuario del servidor SMTP que envía los correos.
SMTP_STARTTLS	true | false	Si el servidor SMTP usa TTL para cifrar la comunicación.
EMAIL_FROM	String	Nombre del emisor del email.
CLAIM_PAGE_URL	String	URL de la claim page.
PAYLOAD_PUBLIC_SERVER_URL	String	URL del propio Dashboard.

6.1.1.4 Variables de Redis

Nombre de la variable	Valor esperado	Para qué sirve
ALLOW_EMPTY_PASSWORD	yes | no	Si se permite el logueo con contraseña vacía.

6.1.1.5 Variables de Mongo

Nombre de la variable	Valor esperado	Para qué sirve
MONGO_INITDB_ROOT_USERNAME	String	El usuario de root para el motor de mongo.
MONGO_INITDB_ROOT_PASSWORD	String	La contraseña del usuario root.

6.1.1.6 Variables de Mongo Express

Nombre de la variable	Valor esperado	Para qué sirve
ME_CONFIG_MONGODB_SERVER	String	Host de la base de datos mongo. Al ser un

		docker-compose, se puede referenciar el nombre del servicio.
ME_CONFIG_MONGODB_PORT	Entero positivo	Puerto del servidor de mongo.
ME_CONFIG_MONGODB_ADMINUSERNAME	String	Username del root de mongo.
ME_CONFIG_MONGODB_ADMINPASSWORD	String	Password del root de mongo.

6.1.2 Verificador

```
services:
 web-verifier-plus:
  container_name: web-verifier-plus
  build:
   context: .
   dockerfile: Dockerfile
  environment:
   NEXT_TELEMETRY_DISABLED: 1
   DB_USER:
   DB_PASS_:
   DB_HOST:
  ports:
   - "{VERIFIER_PORT}:3000"
  depends_on:
   - mongo

 mongo:
  image: mongo
  restart: always
  environment:
   MONGO_INITDB_ROOT_USERNAME:
   MONGO_INITDB_ROOT_PASSWORD:
```

6.1.2.1 Variables del Verificador

Nombre de la variable	Valor esperado	Para qué sirve
NEXT_TELEMETRY_DISABLED	0 | 1	Determina si Vercel, desarrolladora del framework NextJS, toma

		datos anónimos de telemetría sobre la aplicación. 0 para que lo haga; 1 para que no.
DB_USER	String	Username del usuario de mongo.
DB_PASS	String	Password del usuario de mongo.
DB_HOST	String	Host de la base de datos mongo. Al ser un docker-compose, se puede referenciar el nombre del servicio.
VERIFIER_PORT	Entero positivo	El puerto que se expondrá para poder acceder al verificador.

6.1.2.2 Variables de Mongo

Las mismas que para el mongo del emisor.

6.2 Archivos de ejemplo

Aquí se colocarán los archivos docker-stack-example.yml correspondientes a cada servicio desplegado en el servidor de Docker de la UNRN.

6.2.1 Emisor

```
version: '3.8'

networks:
 traefik-unrn:
  external: true

services:
  coordinator:
    image: digitalcredentials/workflow-coordinator:1.0.0
    environment:
      - ENABLE_STATUS_SERVICE=
      - PUBLIC_EXCHANGE_HOST=
      - TENANT_TOKEN_MC_UNRN=
    ports:
      - '4005:4005'
    networks:
      - traefik-unrn
```

```
    deploy:
      mode: replicated
      replicas: 1

  signing:
    image: digitalcredentials/signing-service:1.0.0
    environment:
      - TENANT_SEED_MC_UNRN=
      - TENANT_DIDMETHOD_MC_UNRN=
    networks:
      - traefik-unrn
    ports:
      - '4006:4006'
    deploy:
      mode: replicated
      replicas: 1

  transactions:
    image: digitalcredentials/transaction-service:0.4.0
    networks:
      - traefik-unrn
    deploy:
      mode: replicated
      replicas: 1

  emisor:
    image:
registrygitlab.unrn.edu.ar/lia/microcredenciales/emisor:37d1bd6a

    depends_on:
      - coordinator
      - redis
      - mongo
    environment:
      - COORDINATOR_URL=
      - REDIS_URL=
      - REDIS_PORT=
      - MONGODB_URI=
      - PAYLOAD_SECRET=
      - TENANT_NAME=MC_UNRN
      - SMTP_HOST=
      - SMTP_USER=
      - SMTP_PASS=
      - EMAIL_FROM=
      - CLAIM_PAGE_URL=
      - PAYLOAD_PUBLIC_SERVER_URL=
```

```
    #ports:
    #  - '3000:3000'
    networks:
      - traefik-unrn
#    volumes:
#      - /opt/emisor/src:/home/node/app/src
    deploy:
      mode: replicated
      replicas: 1
      labels:
       - "traefik.enable=true"
       - "traefik.port=3000"
       -
"traefik.http.routers.emisor.rule=Host(`microcredenciales.unrn.edu.ar`)"
       - "traefik.http.routers.emisor.tls=true"
       - "traefik.http.routers.emisor.service=emisor"
       - "traefik.http.services.emisor.loadbalancer.server.port=3000"
       - "traefik.http.routers.emisor.entrypoints=websecure"
       # HTTP
       - "traefik.http.routers.emisor-http.entrypoints=web"
       -
"traefik.http.routers.emisor-http.rule=Host(`microcredenciales.unrn.edu.
ar`)"
       # Redirect
       - "traefik.http.routers.emisor-http.middlewares=emisor-https"
       -
"traefik.http.middlewares.emisor-https.redirectscheme.scheme=https"
  claim-page:
    image: digitalcredentials/admin-dashboard-claim-page:1.0.0
    depends_on:
      - emisor
    ports:
      - '8080:8080'
    networks:
      - traefik-unrn
    deploy:
      mode: replicated
      replicas: 1

  redis:
    image: redis:alpine
    environment:
      - ALLOW_EMPTY_PASSWORD=
    ports:
      - '6379:6379'
    networks:
```

```
      - traefik-unrn
    deploy:
      mode: replicated
      replicas: 1

  mongo:
    image: l33tlamer/mongodb-without-avx:5.0.20
    # image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME:
      MONGO_INITDB_ROOT_PASSWORD:
    ports:
    - "27017:27017"
    networks:
      - traefik-unrn
    volumes:
      - mongodb:/data/db
    deploy:
      mode: replicated
      replicas: 1

  mongo-express:
    image: mongo-express:latest
    ports:
    - "8081:8081"
    networks:
      - traefik-unrn
    environment:
      ME_CONFIG_MONGODB_SERVER:
      ME_CONFIG_MONGODB_PORT:
      ME_CONFIG_MONGODB_ADMINUSERNAME:
      ME_CONFIG_MONGODB_ADMINPASSWORD:
    depends_on:
    - mongo
    deploy:
      mode: replicated
      replicas: 1

volumes:
  mongodb:
```

6.2.2 Verificador

```
version: '3.8'
```

```
networks:
 {VERIFICADOR_NETWORK}:
  external: true

services:
 verificador:
  image: verificador:{TAG}
  environment:
   - NEXT_TELEMETRY_DISABLED=1
   - DB_USER=
   - DB_PASS=
   - DB_HOST=
  networks:
   - {VERIFICADOR_NETWORK}
  deploy:
   mode: replicated
   replicas: 1
   labels:
    - "traefik.enable=true"
    - "traefik.port={VERIFICADOR_PORT}"
    - "traefik.http.routers.verificador.rule=Host(`{VERIFICADOR_URL})"
    - "traefik.http.routers.verificador.tls=true"
    - "traefik.http.routers.verificador.service=verificador"
    -
"traefik.http.services.verificador.loadbalancer.server.port={VERIFICADOR
_PORT}"
    - "traefik.http.routers.verificador.entrypoints=websecure"
    # HTTP
    - "traefik.http.routers.verificador-http.entrypoints=web"
    -
"traefik.http.routers.verificador-http.rule=Host(`VERIFICADOR_URL`)"
    # Redirect
    -
"traefik.http.routers.verificador-http.middlewares=verificador-https"
    -
"traefik.http.middlewares.verificador-https.redirectscheme.scheme=https"
  depends_on:
   - mongo
```

6.2.2.1 Variables del verificador

Nombre de la variable	Valor esperado	Para qué sirve
VERIFICADOR_NETWORK	String	Red interna de Docker para aislar los servicios.

VERIFICADOR_URL	URL	Dominio asignado al servicio en Traefik.
VERIFICADOR_PORT	Entero positivo	Puerto donde el servidor de la aplicación escuchara.
NEXT_TELEMETRY_DISABLED	0 | 1	Determina si Vercel, desarrolladora del framework NextJS, toma datos anónimos de telemetría sobre la aplicación. 0 para que lo haga; 1 para que no.
DB_USER	String	Username del usuario de mongo.
DB_PASS	String	Password del usuario de mongo.
DB_HOST	String	Host de la base de datos mongo. Al ser un docker-compose, se puede referenciar el nombre del servicio.
VERIFIER_PORT	Entero positivo	El puerto que se expondrá para poder acceder al verificador.

6.3 Ejemplos de código

6.3.1 Ejemplo de Plantilla para Credenciales Verificables

```
{
 "type": [
  "VerifiableCredential",
  "EducationalCredential"
 ],
 "proof": {
  "jws": "eyJhbGciOiJFZERTQSJ9..",
  "type": "Ed25519Signature2018",
  "created": "2023-01-01T00:00:00Z",
  "proofPurpose": "assertionMethod",
  "verificationMethod":
"https://campusbimodal.unrn.edu.ar/issuer/keys/1"
 },
 "issuer": {
```

```
  "id": "did:example:76e12ec712ebc6f1c221ebfeb1f",
  "name": "Universidad Nacional de Río Negro"
 },
 "@context": [
  "https://www.w3.org/ns/credentials/v2",
  "https://www.w3.org/ns/credentials/examples/v2"
 ],
 "validFrom": "{{validoDesde}}",
 "credentialStatus": {
  "id": "https://campusbimodal.unrn.edu.ar/status/default",
  "type": "CredentialStatusList2021"
 },
 "credentialSchema": {
  "id": "https://www.w3.org/2018/credentials/examples/v1",
  "type": "JsonSchemaValidator2018"
 },
 "credentialSubject": {
  "type": [
   "AchievementSubject"
  ],
  "name": "{{earnerName}}",
  "achievement": {
   "type": [
    "Achievement"
   ],
   "criteria": {
    "type": "Criteria",
    "narrative": "El alumno {{earnerName}} ha finalizado el
{{degreeType}} en {{subject}} de manera satisfactoria."
   }
  },
  "hasCredential": {
   "id": "https://cred.127.0.0.1.nip.io/api/claim/default",
   "name": "{{credentialName}}",
   "type": [
    "EducationalOccupationalCredential"
   ],
   "logoURL": "{{logoUrl}}",
   "description": "{{description}}",
   "currentYear": "{{currentYear}}",
   "competencyRequired": "Competencias no especificadas.",
   "credentialCategory": "badge"
  }
 }
}
```

6.3.2 Ejemplo de Plantilla para email de notificacion

```
<html>
 <head>
  <style>
   body {
    font-family: 'Arial', sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
   }
   .email-container {
    max-width: 600px;
    margin: 40px auto;
    background-color: #ffffff;
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    border: 3px solid #b30000;
   }
   .header {
    text-align: center;
    padding-bottom: 30px;
   }
   .logo {
    max-width: 200px;
    height: auto;
   }
   .content {
    font-size: 16px;
    color: #333333;
    line-height: 1.8;
   }
   .content p {
    margin: 15px 0;
   }
   .qr-code {
    text-align: center;
    margin: 30px 0;
   }
   .qr-code img {
    max-width: 200px;
    border: 2px solid #b30000;
    border-radius: 10px;
   }
   .footer {
```

```
    text-align: center;
    font-size: 14px;
    color: #666666;
    margin-top: 40px;
    border-top: 1px solid #dddddd;
    padding-top: 20px;
   }
   a {
    color: #b30000;
    text-decoration: underline;
   }
   a:hover {
    color: #a00000;
   }
  </style>
 </head>
 <body>
  <div class="email-container">
   <div class="header">
    <img src="{{logoUrl}}" alt="Logo de la Universidad" class="logo">
   </div>
   <div class="content">
    <p>Estimado/a {{earnerName}},</p>

    <p>¡Felicitaciones por haber completado tu
<strong>{{degreeType}}</strong> en <strong>{{subject}}</strong>! Estamos
muy orgullosos de tu logro.</p>
    <p>Ve a reclamar tu credencial:</p>
   </div>
    <a href="{{link}}">Haz clic aquí</a>
   <div class="footer">
    <p>Si tienes alguna pregunta, por favor contáctanos en <a
href="mailto:support@university.edu">lia@unrn.edu.ar</a>.</p>
    <p>&copy; {{currentYear}} Tu Universidad. Todos los derechos
reservados.</p>
   </div>
  </div>
 </body>
</html>
```