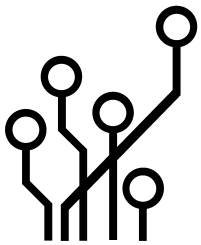


PROJECT ORIGIN

projectorigin2023@gmail.com

Specifica Tecnica

Versione	1.0.0
Responsabile	Beschin Michele
Redattori	Andreetto Alessio, Lotto Riccardo
Verificatori	Ibra Elton
Uso	Esterno
Destinatari	*ProjectOrigin* Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Nel presente documento si fornisce una visione approfondita dell'architettura, del design e delle specifiche tecniche del progetto *Personal Identity Wallet*

Registro delle modifiche

Vers.	Data	Autore	Ruolo	Descrizione
2.0.0	2020-09-24	Beschin Michele	Responsabile	Approvazione documento
0.1.0	2020-09-19	Ibra Elton	Verificatore	Verifica$_g$del documento
0.1.2	2020-09-20	Andreetto Alessio, Lotto Riccardo	Analisti, Amministratori	stesa la struttura definitiva e ampliati tali paragrafi
0.1.1	2023-09-18	Corbu Teodor	Analista	Aggiunte parti Introduzione (capitolo Architettura), Data Scrapper, React Material UI
0.1.0	2023-08-25	Andreetto Alessio	Verificatore	Verifica$_g$del documento
0.0.1	2023-08-24	Corbu Teodor	Analista	Creazione struttura documento, aggiunta Introduzione e spiegazione dei database

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Scopo del prodotto . . . 3
1.3 Note Esplicative . . . 3
1.4 Riferimenti . . . 3

2 Tecnologie Utilizzate **5**
2.1 Tecnologie Back-end . . . 5
2.2 Tecnologie Front-end . . . 5
2.3 Tecnologie Database . . . 5
2.4 Tecnologie Di Supporto . . . 5
2.5 Autenticazione e Sicurezza . . . 5

3 Architettura **7**
3.1 Introduzione . . . 7
3.2 Grafico delle componenti . . . 7
3.2.1 Container **nginx** . . . 7
3.2.2 Containers dell'**Issuer** . . . 7
3.2.3 Containers del **Wallet** . . . 8
3.2.4 Containers del **Verifier** . . . 8
3.2.5 Container **Adminer** . . . 8
3.3 Componenti dell'**Issuer** . . . 9
3.3.1 OriginiIssuerApi (backend) . . . 9
3.3.2 originIssuer (frontend) . . . 10
3.3.3 originIssuerDB (database): . . . 11
3.4 Componenti del **Wallet** . . . 12
3.4.1 OriginWalletApi (backend) . . . 12
3.4.2 originWallet (frontend) . . . 13
3.4.3 originWalletDB (database): . . . 14
3.5 Componenti del **Verifier** . . . 15
3.5.1 originVerifier (frontend) . . . 15
3.6 Design pattern . . . 15
3.6.1 Strategy . . . 15
3.6.2 Il pattern architetturale Model-View-ViewModel . . . 16

4 Elenco dei requisiti **17**
4.1 Qualità . . . 18
4.2 Copertura test . . . 19
4.2.1 Test di Unità . . . 19

1 Introduzione

1.1 Scopo del documento

La Specifica Tecnica si pone come obiettivo di descrivere in modo esaustivo l'organizzazione della struttura del software, delle tecnologie adottate e delle scelte architetturali compiute dal gruppo durante le fasi di progettazione e di codifica del prodotto.
All'interno del documento si possono trovare gli schemi delle classi per delineare l'architettura e le funzionalità chiave del prodotto, con l'obiettivo di fornire una comprensione completa e chiara del sistema e delle interazioni interne.
Il documento contiene anche una sezione per i requisiti che vengono soddisfatti dal prodotto; questo permette al gruppo di valutare il progresso del lavoro e di tener traccia degli obiettivi imposti.

1.2 Scopo del prodotto

Lo scopo del prodotto è quello di creare una versione semplificata di un applicativo per implementare e rilasciare un "portafoglio di identità digitale" conforme a un insieme di standard, in modo che possa essere utilizzato con qualsiasi servizio, che adotti tale struttura, in qualsiasi paese dell'UE.
In particolare, si dovrà realizzare una web app$_g$ avendo queste componenti architetturali:

- Un componente back-office per consentire al dipendente dell'organizzazione emittente di verificare$_g$ manualmente la richiesta di credenziali e autorizzarne l'emissione;
- Un componente di interazione con l'utente dimostrativo per consentire all'utente (titolare) di navigare e richiedere specifiche credenziali da un emittente (ad esempio, il sito di una demo universitaria);
- Un componente di interazione con l'utente dimostrativo per consentire all'utente (titolare) di navigare un sito verificatore$_g$ e fornire le credenziali richieste;
- Un'app front-end per l'utente per archiviare e gestire le proprie credenziali;
- Un componente di comunicazione per consentire lo scambio di credenziali/presentazioni secondo un protocollo standard - il componente di comunicazione sarà implementato tre volte nei tre contesti (lato emittente, lato titolare, lato verificatore).

1.3 Note Esplicative

Alcuni termini utilizzati nel documento possono avere significati ambigui a seconda del contesto. Al fine di evitare equivoci, è stato creato un *Glossario v.2.0.0* contenente tali termini e il loro significato specifico. Per segnalare che un termine è presente nel *Glossario v.2.0.0* , sarà aggiunta una "g" a pedice accanto al termine corrispondente nel testo.

1.4 Riferimenti

1. Normativi:

- ***Norme di Progetto v.2.0.0*** : contengono le norme e gli strumenti per gli analisti;
- **Capitolato d'appalto C3**: https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C3.pdf;
- **Regolamento del progetto didattico:**: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/PD02.pdf.

2. Informativi:

- **Analisi dei Requisiti v1.0.0**;

- **Qualità di prodotto – slide T8 di Ingegneria del Software:** : https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T08.pdf;
- **Qualità di processo – slide T9 di Ingegneria del Software:** : https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T09.pdf;
- **Verifica e Validazione: introduzione – slide T10 di Ingeneria del Software:**: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T10.pdf;
- **Verifica e Validazione: introduzione – slide T11 di Ingeneria del Software:**: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T11.pdf;
- **Verifica e Validazione: introduzione – slide T12 di Ingeneria del Software:**: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T12.pdf.

2 Tecnologie Utilizzate

Essendo la struttura delle webapp a microservizi ono state utilizzate le segueni tecnologie:

- Docker,
- Docker Container.

Per il deployment dei container è stata gestita la configurazione tramite un file docker compose *docker-compose.yaml*.

2.1 Tecnologie Back-end

- Node.js + Javascript
- Express: per la creazione di endpoint e creazione di server Http.

2.2 Tecnologie Front-end

Per la realizzazione della parte grafica degli applicativi web sono state usate le seguenti tecnologie:

- Vite + React + Javascript,
- Axios per eseguire le chiamate http,
- Material UI.

Vite nello specifico ci è servito per gestire meglio le dipendenze, fare il building della webapplication in maniera migliore e avere più velocità.

Material-UI è una libreria di componenti UI (User Interface) per React fatto da Google. Material-UI è una libreria che viene utilizzata principalmente per realizzare interfacce utente dinamiche e personalizzate. Segue il design system "Material Design", che si occupa di fornire linee guida di design per la realizzazione di applicazioni e siti web.

2.3 Tecnologie Database

- DBMS PostegreSQL

2.4 Tecnologie Di Supporto

- nginx: Proxy manager per gestire il flusso di dati in arrivo dall'esterno delle nostre componenti e reindirizzarlo all'interno dei nostri componenti docker.
- WaltID: utilizzato per rispettare lo standard OpenIDV4 CI-VP$_g$con container messi a disposizione da waltID.

2.5 Autenticazione e Sicurezza

Per la realizzazione dell'autenticazione della webpp originIssuer e originWallet si è optato per un processo di autenticazione tramite username e password. Questi 2 campi sono memorizzati all'interno del database. Per quanto rigarda la password viene memorizzata criptata in particolare solamente l'hash$_g$insieme ad un salt$_g$. Queste componenti insieme allo username servono per procedere con il processo di autenticazione. Tuttavia in caso di furto e dump del database non sarà comunque possibile risalire ai dati di accesso. La password viene criptata tramite un salt generato in maniera casuale. Al momento di login, fornendo username e password, fornisco anche il salt presente nel database; l'hash risultante verrà confrontato con l'hash nel database. Questo confronto tra hash e non password in chiaro ne garantisce la sicurezza.

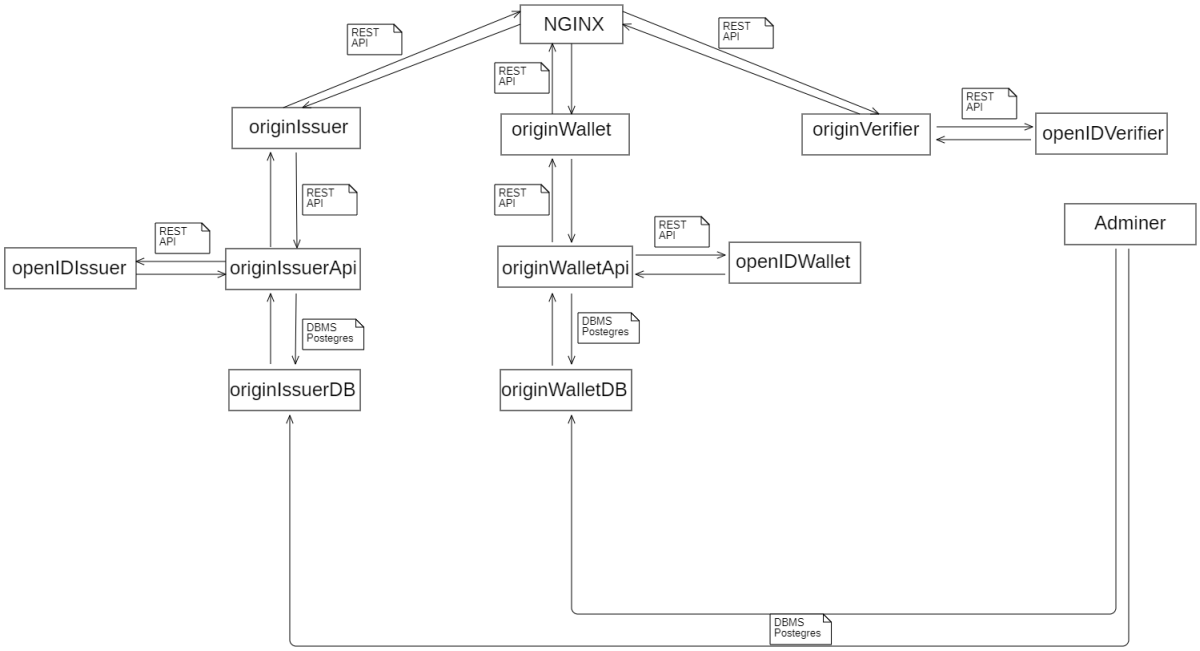
L'autenticazione e l'acceso riervato da parte del frontend da parte del backend è stato realizzato tramite token JWT_g. Questi token oltre che permettere l'accesso a risorse riservate a coloro che hanno i permessi, consentono di memorizzare nel token stesso variabili dell'utente. La decriptazione del token viene fatta nel backend tramite secret messa da configurazione. Inoltre sempre da configurazione è stata implementata la validità temporale del token. Nel caso del Wallet inoltre è stata aggiunta la funzionalità del refresh del token occasionale. Questo permette, nel caso in cui ci fosse un furto del token tramite un attacco man in the middle, di rendere inutilizzabile il token dato che verrebbe rigenerato. La rigenerazione avviene ogni qualvolta il frontend abbia un interazione con il backend.

3 Architettura

3.1 Introduzione

Per la Realizzazione delle 3 webapp è stata adottata un architettura a microservizi, separando le funzioni di back-end da quelle del front-end. È stato fatto il deploy dei vari microservizi di ciascuna webapp su container docker differenti.

3.2 Grafico delle componenti



3.2.1 Container nginx

Lo utilizziamo come server proxy$_g$per gestire il reindirezzamento del traffico http tramite domini verso i container interni della rete docker. Alcuni container, nello specifico quelli addibiti allo standard OpenId, utilizzati da noi e sviluppati da WaltId, per funzionare la loro configurazione è basata solo sull'utilizzo di domini. Pertando questo necessita forzatamente l'utilizzo dei precedementemente citati domini per accedere alle funzionalità.

3.2.2 Containers dell'Issuer

- **originIssuerApi:** È la componente di back-end della webapp Issuer che si occupa di gestire le richieste provenienti dal front-end e di comunicare con il database *originIssuerDB* per la memorizzazione dei dati.
- **originIssuer:** È la componete di front-end della webapp Issuer. Essa si interfaccia con l'utente per inviare segnali e ricevere dati dal Back-end *originIssuerApi*. Per la realizzazione del codice è stato adottato il pattern architetturale MVVM (Model-View-ViewModel).
- **originIssuerDB:** È la componente della webapp Issuer che va a eseguire operazioni sul database per la richiesta e la memorizzazione di dati.
- **openIdIssuer:** È una componente della libreria WaltID per mantenere lo standard openId di comunicazione tra Issuer e le altre webapp, al fine di rispettare le richieste del capitolato.

3.2.3 Containers del Wallet

- **originWalletApi:** È la componente di back-end della webapp Wallet che si occupa di gestire le richieste provenienti dal front-end e di comunicare con il database *originWalletDB* per la memorizzazione dei dati.
- **originWallet:** È la componete di front-end della webapp Wallet. Essa si interfaccia con l'utente per inviare segnali e ricevere dati dal Back-end *originWalletApi*. Per la realizzazione del codice è stato adottato il pattern architetturale MVVM (Model-View-ViewModel).
- **originWalletDB:** È la componente della webapp Wallet che va a eseguire operazioni sul database per la richiesta e la memorizzazione di dati.
- **openIdWallet:** È una componente della libreria WaltID per mantenere lo standard openId di comunicazione tra Wallet e le altre webapp, al fine di rispettare le richieste del capitolato.

3.2.4 Containers del Verifier

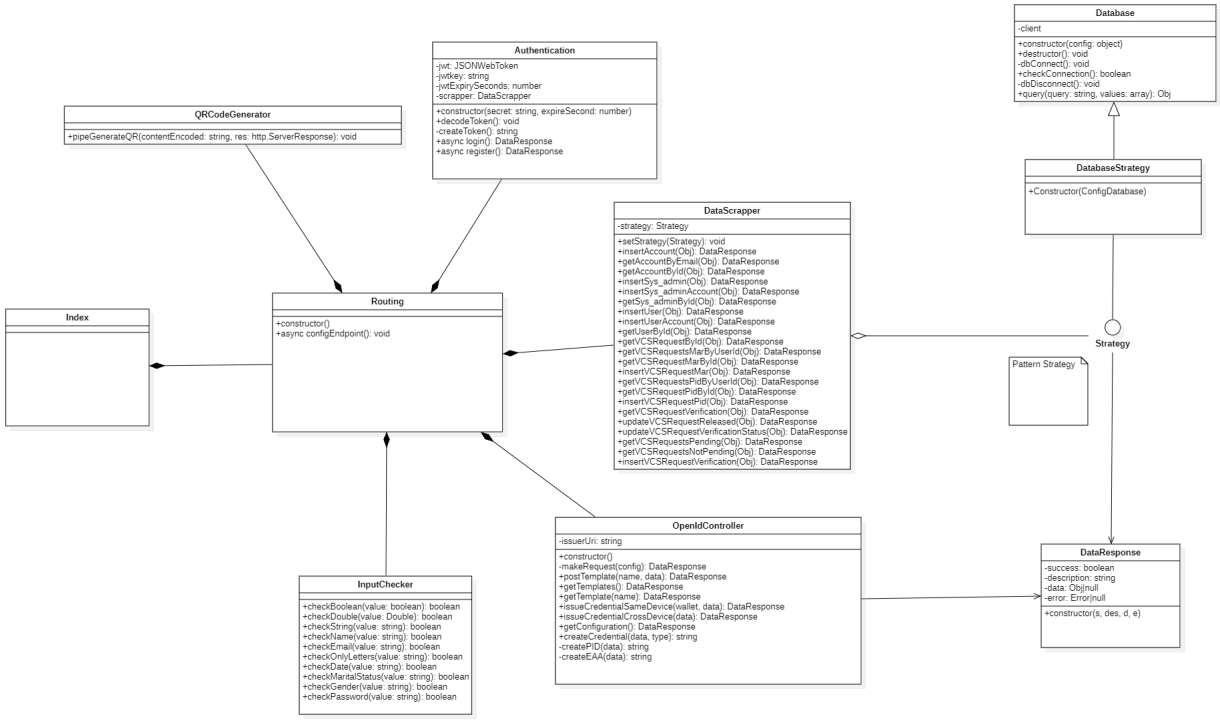
- **originVerifier:**È la componete di front-end della webapp Verifier. Essa si interfaccia con l'utente per inviare segnali e ricevere dati dal Back-end *originVerfierApi*. Per la realizzazione del codice è stato adottato il pattern architetturale MVVM (Model-View-ViewModel).
- **openIdVerifier:**È una componente della libreria WaltID per mantenere lo standard openId di comunicazione tra Verifier e le altre webapp, al fine di rispettare le richieste del capitolato.

3.2.5 Container Adminer

È un container che permette agli sviluppatori di gestire i database tramite interfaccia web.

3.3 Componenti dell'Issuer

3.3.1 OriginiIssuerApi (backend)

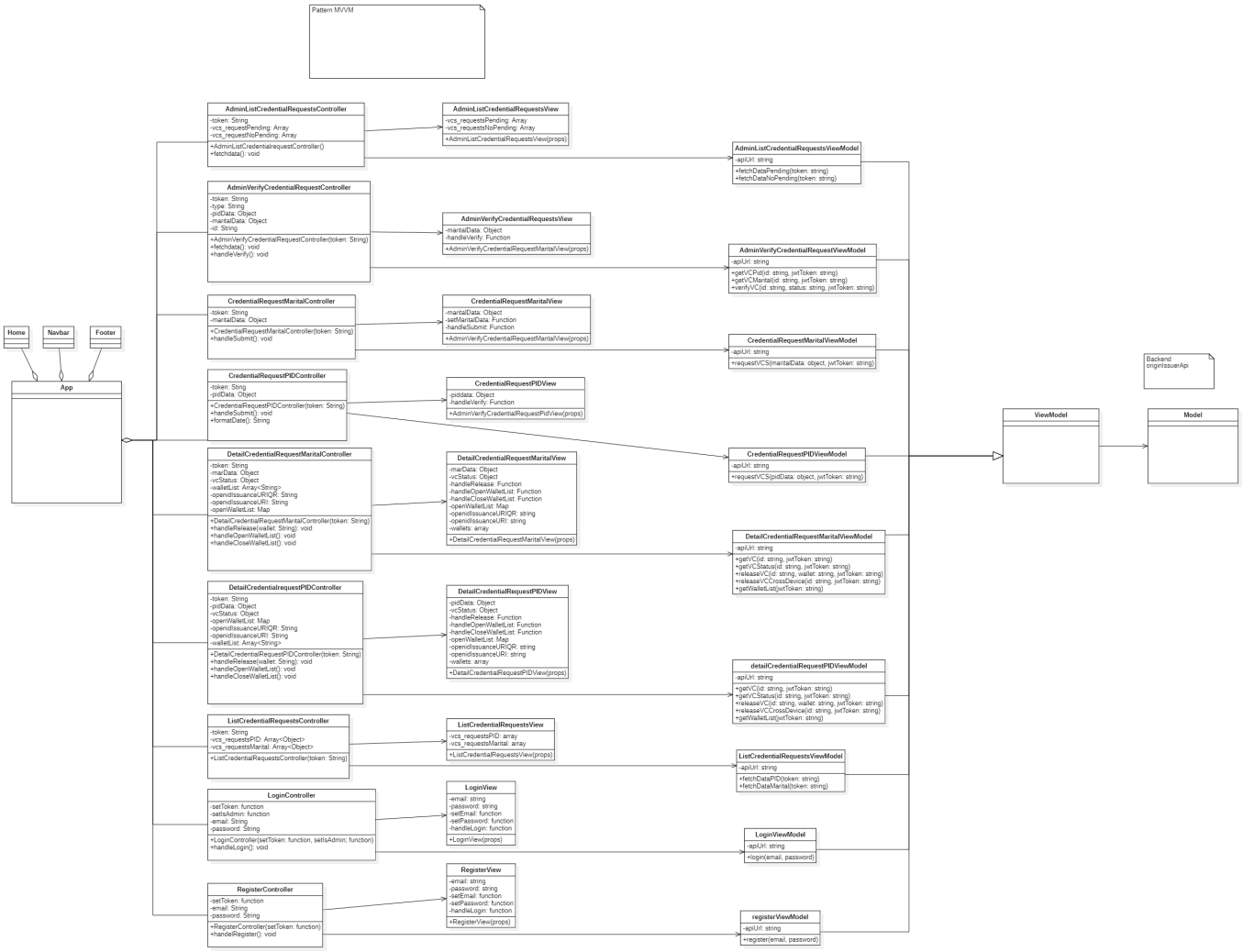


- **authentication:**È una classe che si occupa della autenticazione, ovvero: login, registrazione.
- **datasScrapper:**È un insieme di classi che si occupa di dialogare con il database reperendo e memorizzando i dati da esso. È stato realizzato tramite un insieme di 3 classi che rispettano il pattern strategy.
- **openid:**È la classe che si occupa di comunicare con il container openIdIssuer. Essendo le chiamate dipendenti strettamente dallo sviluppo della libreria waltId, il gruppo ha preferito far comunicare il front-end non direttamente con la libreria waltId ma dialogare con questo layer intermedio. In questa maniera il backend si occuperà di fare le chiamate con la libreria waltId del container openIdIssuer.
- **DataResponse:**È una classe che si occupa di parametrizzare il tipo di ritorno che il backend fornisce.
- **InputChecker:**Contiene dei metodi necessari per la verifica e la correttezza degli input fatti al backend.
- **QRCodeGenerator:**È una classe fondamentale nel processo di credential issuing. Vi sono 2 modi per generare una credenziale ovvero: cros device e same device. Questa classe si occupa di generare un qrcode senza generare direttamente l'immagine ma dando via risorsa al frontend lo stream dell'immagine del qrcode tramite il metodo *pipeGenerateQR*.
- **Routing:** In questa classe vengono definite 2 funzionalità principali:
  - Cors ovvero una funzionalità per limitare l'uso da altri dispositivi. Questa opzione viene configuarata tramite corsOptions specificando gli indirizzi di origine da cui un utilizzatore potrà usare il backend.
  - Express è una funzionalità che permette di offrire degli endpoint$_g$con cui un altro applicativo potrà fare delle chiamate http.

Nel componente vengono configurate le nostre classi precedentemente elencate. Inoltre la clsse specifica tutte le chiamate possibili del backend originIssuerApi.

- **index:**Viene creato il routing e il metodo di configurazione degli endpoint.

3.3.2 originIssuer (frontend)

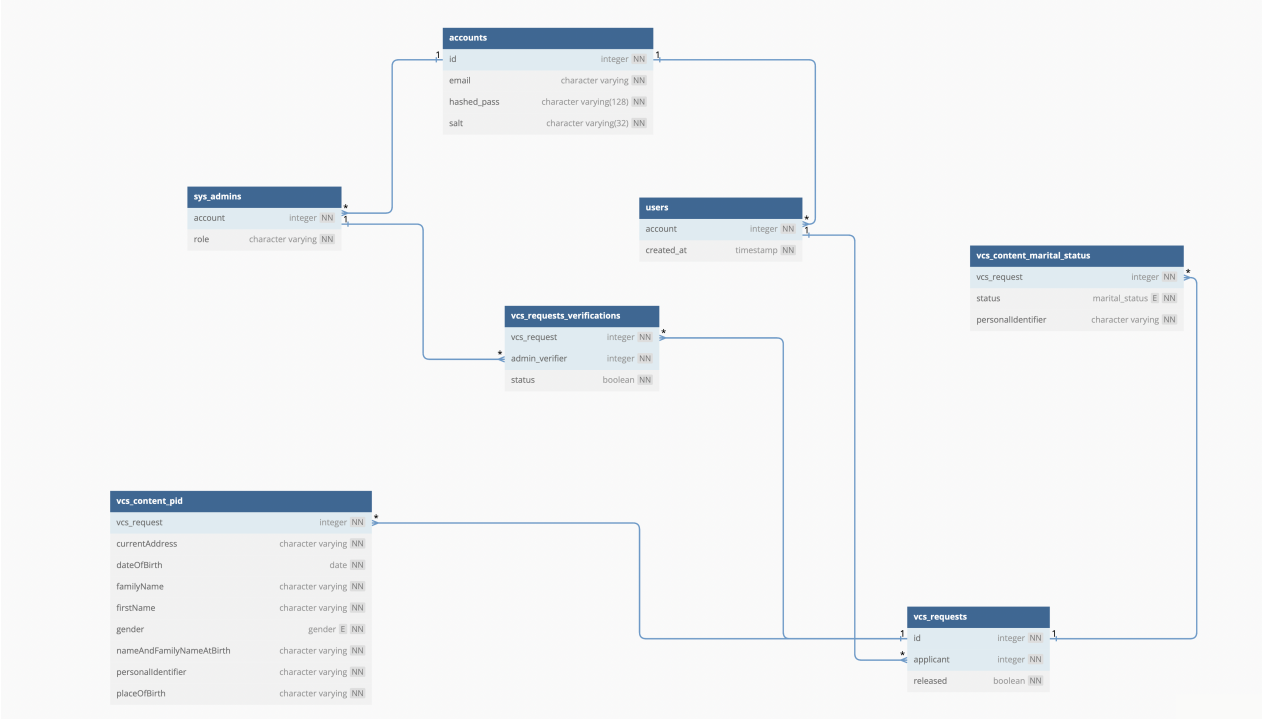


- **components/Navbar:** questa è la componente che definisce la *navbar* dell'applicazione, che differisce dal tipo di utente che è loggato (admin, user, guest).
- **components/useToken:** componente che gestisce la memorizzazione e l'eliminazione del token. Tramite il suo utilizzo si riesce a garantire l'autenticazione del front-end presso il back-end.
- **components/Home:** componente che rappresenta la pagina iniziale dell'applicazione.
- **components/LicenseLable:** componente che gestische i dettagli sulla licenza dell'applicazione.
- **components/Logout:** componente che gestisce il logout dell'utente.

Ogni seguente componente è composta da 3 parti: *controller*, *viewmodel* e *view*. Il controller crea la corrispondente viewModel e view. Il controller si occupa di gestire gli eventi provenienti dalla view e di aggiornare la viewModel, che a sua volta aggiorna il model presente nel back-end. La view si occupa di mostrare i dati presenti nella viewModel e di gestire gli eventi provenienti dall'utente.

- **Login:** questa è la componente che gestisce la pagina di login dell'applicazione.
- **Register:** questa è la componente che gestisce la pagina di registrazione dell'applicazione.
- **CredentialRequestPID:** questa è la componente che gestisce la pagina di richiesta di una credenziale PID.
- **CredentialRequestMarital:** questa è la componente che gestisce la pagina di richiesta di una credenziale Marital.
- **ListCredentialReqauest:** questa è la componente che gestisce la pagina di visualizzazione delle richieste di credenziali. Si viene reindirizzati a questa pagina dopo aver fatto richiesta di una credenziale.
- **DetailCredentialRequestPID:** questa è la componente che gestisce la pagina di dettaglio di una richiesta di credenziale PID.
- **DetailCredentialRequestMarital:** questa è la componente che gestisce la pagina di dettaglio di una richiesta di credenziale Marital.
- **AdminListCredentialRequest:** questa è la componente che gestisce la pagina di visualizzazione delle richieste di credenziali da parte degli admin. In essa sono presenti due liste di richieste: quelle da approvare e quelle già approvate.
- **AdminVerifyCredentialRequest:** questa è la componente che gestisce la pagina di dettaglio di una richiesta di credenziale da parte degli admin.

3.3.3 originIssuerDB (database):



L'immagine sopra riportata descrive il database "issuerdb" implementato mediante un diagramma logico. Issuerdb è stato pensato per gestire e conservare le informazioni legate agli utenti, alle richieste di certificati digitali (VCS requests) e alle verifiche dei certificati stessi. Per quanto richiesto dal capitolato, e

per rispettare la logica dietro tutto il meccanismo dell'Issuer, si distinguono 2 tipi diversi di account: gli account "sys_admins" e gli account "users".

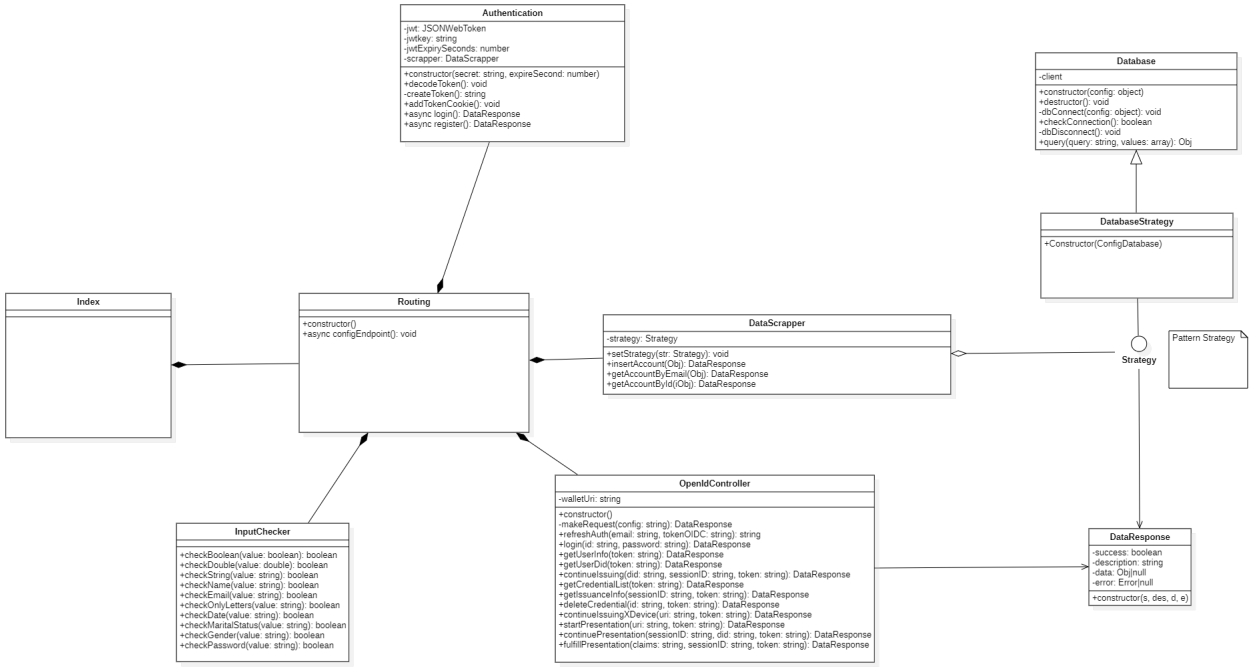
- "sys_admins" sono gli account utilizzati dagli amministratori di sistema, cioè quelle entità che si occupano di approvare (o meglio, verificare) le richieste degli "users" (VCS requests).

- "users", invece, sono gli account utilizzati dai semplici utilizzatori del servizio. Si occupano semplicemente di effettuare delle richieste di certificati di loro interesse alle entità che si occupano di verificare i certificati.

Il contenuto della richiesta approvata e rilasciata può essere di 2 tipi soltanto: un "vcs_content_marital_status" (contenuto riferito allo stato di matrimonio di un utente) o un "vcs_content_pid" (contenuto riferito ad un documento PID di un utente).

3.4 Componenti del Wallet

3.4.1 OriginWalletApi (backend)

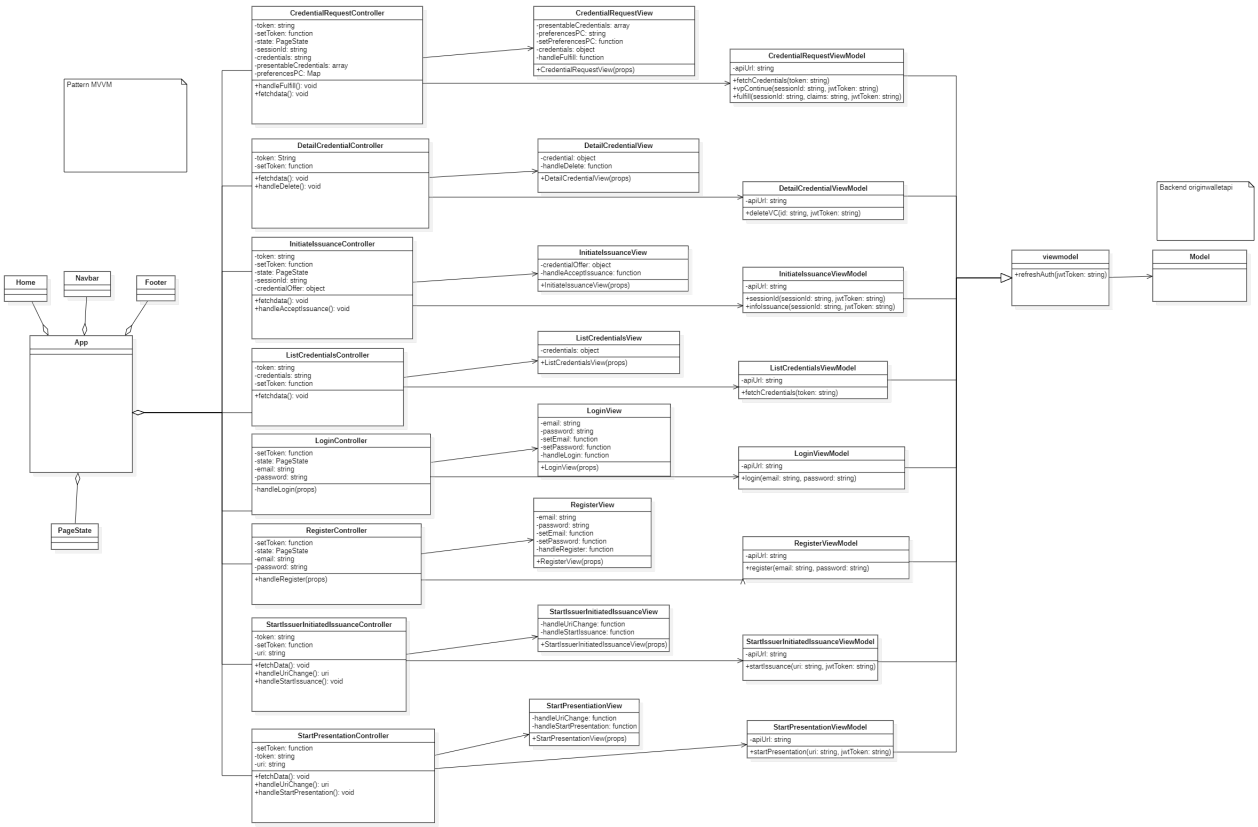


- **authentication:**È una classe che si occupa della autenticazione, ovvero: login, registrazione.
- **datasScrapper:**È un insieme di classi che si occupa di dialogare con il database reperendo e memorizzando i dati da esso. È stato realizzato tramite un insieme di 3 classi che rispettano il pattern strategy. Essendo il database del Wallet irrisorio si potrebbe usare un ulteriore strategia, memorizzando i dati su un file XML o JSON. Questo poichè la struttura dei dati e la complessità è semplice e senza relazioni particolari.
- **openid:**È la classe che si occupa di comunicare con il container openIdWallet. Essendo le chiamate dipendenti strettamente dallo sviluppo della libreria waltId, il gruppo ha preferito far comunicare il front-end non direttamente con la libreria waltId ma dialogare con questo layer intermedio. In questa maniera il backend si occuperà di fare le chiamate con la libreria waltId del container openIdWallet.

- **DataResponse:**È una classe che si occupa di parametrizzare il tipo di ritorno che il backend fornisce.
- **InputChecker:**Contiene dei metodi necessari per la verifica e la correttezza degli input fatti al backend.
- **Routing:** In questa classe vengono definite 2 funzionalità principali:
  - Cors ovvero una funzionalità per limitare l'uso da altri dispositivi. Questa opzione viene configuarata tramite corsOptions specificando gli indirizzi di origine da cui un utilizzatore potrà usare il backend.
  - Express è una funzionalità che permette di offrire degli endpoint con cui un altro applicativo potrà fare delle chiamate http.

  Nel componente vengono configurate le nostre classi precedentemente elencate. Inoltre la clsse specifica tutte le chiamate possibili del backend originIssuerApi.

3.4.2 originWallet (frontend)



- **App:** questa è la pagina iniziale dell'applicazione, dove viene definito il *routing* delle pagine.
- **components/Navbar:** questa è la componente che definisce la *navbar* dell'applicazione, che differisce dal tipo di utente che è loggato (user, guest).
- **components/useToken:** componente che gestisce la memorizzazione e l'eliminazione del token. Tramite il suo utilizzo si riesce a garantire l'autenticazione del front-end presso il back-end.
- **components/Home:** componente che rappresenta la pagina iniziale dell'applicazione.

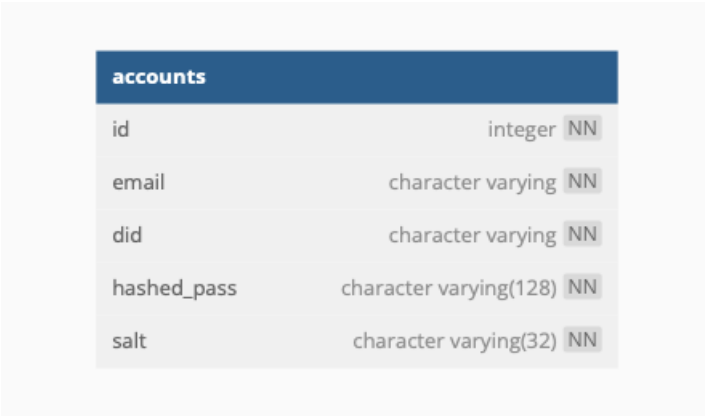
- **components/LicenseLable:** componente che gestische i dettagli sulla licenza dell'applicazione.
- **components/Logout:** componente che gestisce il logout dell'utente.
- **pageState:** componente che gestisce il dialogo ed il reindirizzamento tra le applicazioni. Questa componente memorizza lo stato in cui si trova il *wallet* prima di un reindirizzamento e lo riprende dopo che questo è terminato.

Ogni seguente componente è composta da 3 parti: *controller*, *viewmodel* e *view*. Il controller crea la corrispondente viewModel e view. Il controller si occupa di gestire gli eventi provenienti dalla view e di aggiornare la viewModel, che a sua volta aggiorna il model presente nel back-end. La view si occupa di mostrare i dati presenti nella viewModel e di gestire gli eventi provenienti dall'utente.

- **Login:** questa è la componente che gestisce la pagina di login dell'applicazione.
- **Register:** questa è la componente che gestisce la pagina di registrazione dell'applicazione.
- **ViewModel:** componente che si occupa del collegamento con il back-end, ha quindi un riferimento al *model*. È unico per tutta l'applicazione.
- **ListCredential:** componente che si occupa di mostrare la lista delle credenziali dell'utente presenti nel *Wallet*. Da qui si può andare nel dettaglio di una singola credenziale.
- **DetailCredential:** componente che si occupa di mostrare i dettagli di una credenziale presente nel *Wallet*. Da qui si può eliminare una credenziale.

  NB. Le seguenti componenti hanno nomi controintuitivi ma sono imposti dallo standard *openID*.
- **InitiateIssuance:** componete che si occupa dell'accettazione di una richiesta di credenziale da parte di un *Issuer* e vengo reindirizzato alla pagina *ListCredential*.
- **CredentialRequest:** componente che si occupa del re indirizzamento di una richiesta di presentazione di una credenziale parte del *Verifier*.
- **StartIssuerInitiatedIssuance:** componente necessaria per il *credential issuing* cross-device, quindi attraverso URI openID.
- **StartPresentation:** componente necessaria per la *verifiable presentation* cross-device, quindi fare il fullfill di una presentazione attraverso URI openID.

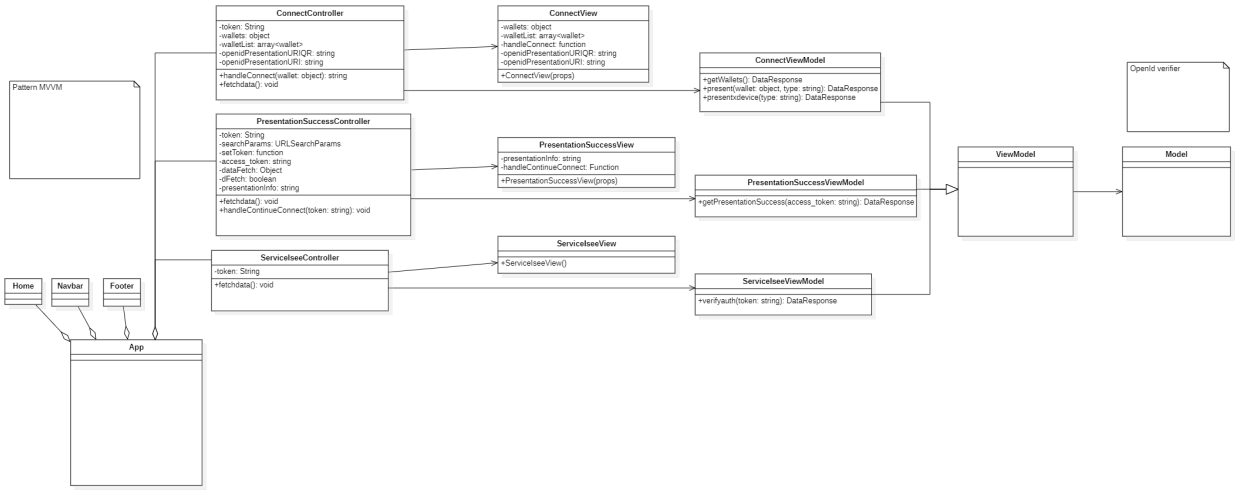
3.4.3 originWalletDB (database):



L'immagine sopra riportata descrive il database "walletdb" implementato mediante un diagramma logico. Walletdb è stato pensato per gestire e conservare (fare lo "storing") le informazioni legate alle credenziali degli utenti, come espresso da capitolato.

3.5 Componenti del Verifier

3.5.1 originVerifier (frontend)



- **App:** questa è la pagina iniziale dell'applicazione, dove viene definito il *routing* delle pagine.
- **components/Navbar:** questa è la componente che definisce la *navbar* dell'applicazione, che differisce dal tipo di utente che è loggato.
- **components/useToken:** componente che gestisce la memorizzazione e l'eliminazione del token. Tramite il suo utilizzo si riesce a garantire l'autenticazione del front-end presso il back-end.
- **components/Home:** componente che rappresenta la pagina iniziale dell'applicazione.
- **components/LicenseLable:** componente che gestische i dettagli sulla licenza dell'applicazione.
- **components/Logout:** componente che gestisce il logout dell'utente.

Ogni seguente componente è composta da 3 parti: *controller*, *viewmodel* e *view*. Il controller crea la corrispondente viewModel e view. Il controller si occupa di gestire gli eventi provenienti dalla view e di aggiornare la viewModel, che a sua volta aggiorna il model presente nel back-end. La view si occupa di mostrare i dati presenti nella viewModel e di gestire gli eventi provenienti dall'utente.

- Connect: componente necessaria per effettuare la connesione e presentare una credenziale di tipo PID al *Verifier* attraverso il proprio *Wallet*.
- PresentationSuccess: componente che si occupa di mostrare il successo della presentazione di una credenziale.
- ServiceIsee: servizio offerto dal *Verifier* che simula la richiesta di un ISEE, per acceder a questo componente si è vincolati all'utilizzo di un token di autorizzazione rilasciato dal componente openID e rilasciato solo in nel caso la procedura di verifiable presentation è completata con successo.

3.6 Design pattern

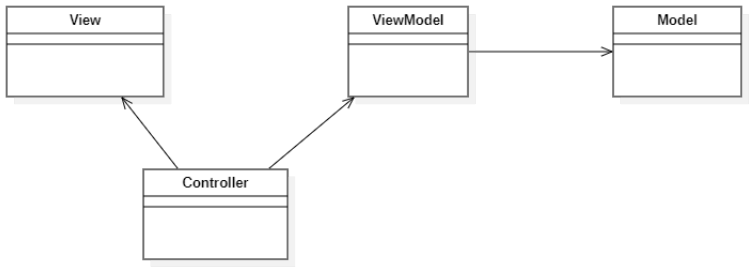
3.6.1 Strategy

Il Pattern Strategy è un design pattern comportamentale che consente ad un oggetto client di scegliere un algoritmo specifico tra una serie di diversia algoritmi senza dover necessariamente modificare il codice sorgente. Questo favorisce la modularità, la flessibilità del codice e l'incapsulamento. Esso è stato utilizzato per la componente del Wallet e dell'Issuer. Essendo che entrambe hanno la necessità, in quanto

web application, di collegarsi al backend. Quest'ultimo deve memorizzare, richiedere e visualizzare i dati in un database. Il gruppo ha deciso di adottare tale pattern data la possibilità in futuro di cambiare sistema di dbms o struttura di contenimento di dati (XML o JSON) cambiando soltanto algoritmo senza dover cambiare il codice sorgente per intero. Questo è stato possibile attraverso l'utilizzo di 3 classi: *dataScapper* essendo l'interfaccia principale con la quale la web application interagiasce, tramite il metodo setStrategy è stato utilizzata la strategia *DatabaseStrategy*. Quest'ultima è una strategia di reperimento dei dati specifici, applicata per una tipologia specifica di database; ereditata dalla classe *Database*. Il pattern utilizzato si discosta da quello imparato durante il corso poichè il linquaggio di programmazione Javascript non implementa l'ereditarietà nelle classi. Per sopperire a tale mancanza è stata usata la relazione *has-a*. Nel Nostro caso ad esempio essendo il database di Wallet semplice e con nessuna relazione è possibile in futuro applicare una strategia JSONStrategy al posto di DatabaseStrategy.

3.6.2 Il pattern architetturale Model-View-ViewModel

È stato utilizzato per separare la parte di business logic dell'applicazione da quella della'ui per rendere più manutenibile il codice e per mantenere una gestione più chiara dei dati e dell'interfaccia utente.
In particolare abbiamo applicato il pattern secondo la seguente struttura :



Introducendo la componente controller all'interno del pattern abbiamo un interazione in più tra la view e il ViewModel. Questo ci permette di:

- Riusciamo a gestire meglio gli handler, ovvero le funzioni chiamate;
- riusciamo meglio a gestire gli eventi;
- riusciamo meglio a gestire i dati ed elaborarli.

Nello specifico quset'ultimo punto era necessario poichè non dovevamo soltanto mostrare dei dati presi direttamnete dal database. In questa maniera abbiamo usato il controller come component di react. Questa struttura è stata adottata per tutte e 3 le web application.

4 Elenco dei requisiti

Requisiti funzionali soddisfatti

Codice	Descrizione	Riferimento	Stato
RF01-O	L'utente inserisce le credenziali nel portale del wallet per iscriversi	UC1	soddisfatto
RF02-O	L'utente visualizza un messaggio di errore per dati immessi non corretti, non risulta registrato al wallet	UC4	soddisfatto
RF03-O	L'utente può accedere al portale wallet attraverso le credenziali di accesso	UC2	soddisfatto
RF04-O	L'utente visualizza un messaggio di errore per credenziali sbagliate al login	UC4	soddisfatto
RF05-O	L'utente può eseguire il logout dal portale wallet	UC3	soddisfatto
RF06-O	L'utente inserisce le credenziali nel portale Issuer sistema per poter iscriversi	UC5	soddisfatto
RF07-O	L'utente visualizza un messaggio di errore durante la registrazione nel portale Issuer sistema per dati immessi non corretti.	UC8	soddisfatto
RF08-O	L'utente visualizza un messaggio di errore durante il login nel portale Issuer sistema per dati immessi non corretti.	UC8	soddisfatto
RF09-O	L'utente può accedere attraverso le credenziali al portale dell'Issuer sistema	UC6	soddisfatto
RF10-O	L'utente può eseguire il logout dal portale dell'Issuer sistema	UC7	soddisfatto
RF11-O	L'utente richiede una credenziale PID identificativa nella portale dell'Issuer sistema	UC10	soddisfatto
RF12-O	L'utente richiede una credenziale EAA nel portale dell'Issuer sistema	UC11	soddisfatto
RF13-O	L'issuer admin effettua il login con le proprie credenziali speciali al portale Issuer sistema	UC6	soddisfatto
RF14-O	L'issuer admin accede alla propria dashboard amministrativa	UC6	soddisfatto
RF15-O	L'issuer admin esamina le richiesta di credenziale nel portale issuer sistema	UC15	soddisfatto
RF16-O	L'issuer admin approva o rifiuta la richiesta di credenziale credenziale portale issuer sistema	UC15	soddisfatto
RF17-O	Se la richiesta di credenziale è approvata, l'issuer sistema genera credenziale richiesta	UC15	soddisfatto
RF18-O	L'holder può verificare nel portale Issuer sistema lo stato della richiesta credenziale	UC12	soddisfatto
RF19-O	Data una richiesta approvata e una credenziale generata l'utente può ottenere nel proprio wallet tale credenziale dal portale issuer sistema	UC14	soddisfatto
RF20-O	L'utente ottiene correttamente la credenziale nel propio wallet	UC14	soddisfatto

RF21-O	L'utente visualizza un errore sul wallet che notifica l'errore di rilascio della credenziale	UC13	soddisfatto
RF22-O	L'utente visualizza una lista di credenziali memorizzate all'interno del proprio wallet	UC16	soddisfatto
RF23-O	L'utente all'interno della propria portale wallet visualizza dettagliatamente la credenziale identificativa PID	UC18	soddisfatto
RF24-O	L'utente all'interno della propria portale wallet visualizza dettagliatamente la credenziale identificativa EAA	UC19	soddisfatto
RF25-O	L'utente elimina le credenziali memorizzate nel wallet	UC20	soddisfatto
RF26-O	Il verifier richiede all'utente una credenziale presente sul wallet personale da verificare	UC21	soddisfatto
RF27-O	L'utente fornisce tramite il proprio wallet una credenziale al verifier da verificare	UC22	soddisfatto
RF28-O	L'utente riesce a visualizzare un messaggio di errore nella piattaforma verifier che notifica l'errore di verifica	UC23	soddisfatto

Numero di requisiti funzionali obbligatori soddisfatti: 28/28.

4.1 Qualità

Requisiti di qualità

Codice	Descrizione	Stato
RQ01-O	Le webapp devono essere sviluppate secondo le regole imposte e descritte nel documento *Norme di Progetto v.2.0.0*	soddisfatto
RQ02-O	Devono essere prodotti e sviluppati dei test sulle funzionalità dei servizi, che assicurano almeno l'80% di copertura del codice prodotto	soddisfatto
RQ03-O	Deve essere redato un documento con gon gli eventuali problemi sorti e le possibili soluzioni per risolverli	soddisfatto
RQ04-O	Deve essere prodotto un documentato con tutte le scelte progettuali e implementative	soddisfatto

Numero di requisiti qualitativi obbligatori soddisfatti: 4/4.

4.2 Copertura test

4.2.1 Test di Unità

- Branch Coverage: 81%
- Statement Coverage: 83%
- Function Coverage: 88%
- Line Coverage: 85%

Test di Integrazione

- Branch Coverage: 91%
- Statement Coverage: 95%
- Function Coverage: 92%
- Line Coverage: 96%