IW – Architettura di dettaglio

Autore: Simone Ballarin

Data: 25/06/18

Destinatari: Athesys

Diario delle modifiche

Data	Descrizione	Autore
03/07/18	Creazione documento. Stesura capitolo DataAccess layer. Inserite descrizioni classe di comunicazione Rest e Blockchain.	Simone Ballarin
04/07/18	Completamento capitolo DataAccess layer. Inizio stesura Business Logic Layer.	Simone Ballarin
05/07/18	Fine stesura Business Logic Layer. Aggiunta metodo userCreationRequest(userModel) in Comunicator. Aggiunta metodo sdkServiceList() in Comunicator. Aggiunti metodi addPII e removePII in DBDao. Inserito diagramma BusinessLogic. Modificato diagramma DataAccess. Stesura del capitole VM Layout. Inseriti capitoli Scopo Documento, Sintesi del documento, Riferimenti normativi. Inserita immagine architettura totale.	Simone Ballarin

Scopo del documento

Il seguente documento *IW – Architettura di dettaglio* ha lo scopo di presentare in maniera dettagliato ogni classe presentata in *IW – Architettura di massima*. Il documento deve fornire una descrizione dei metodi e campi dati presenti.

Sintesi del documento

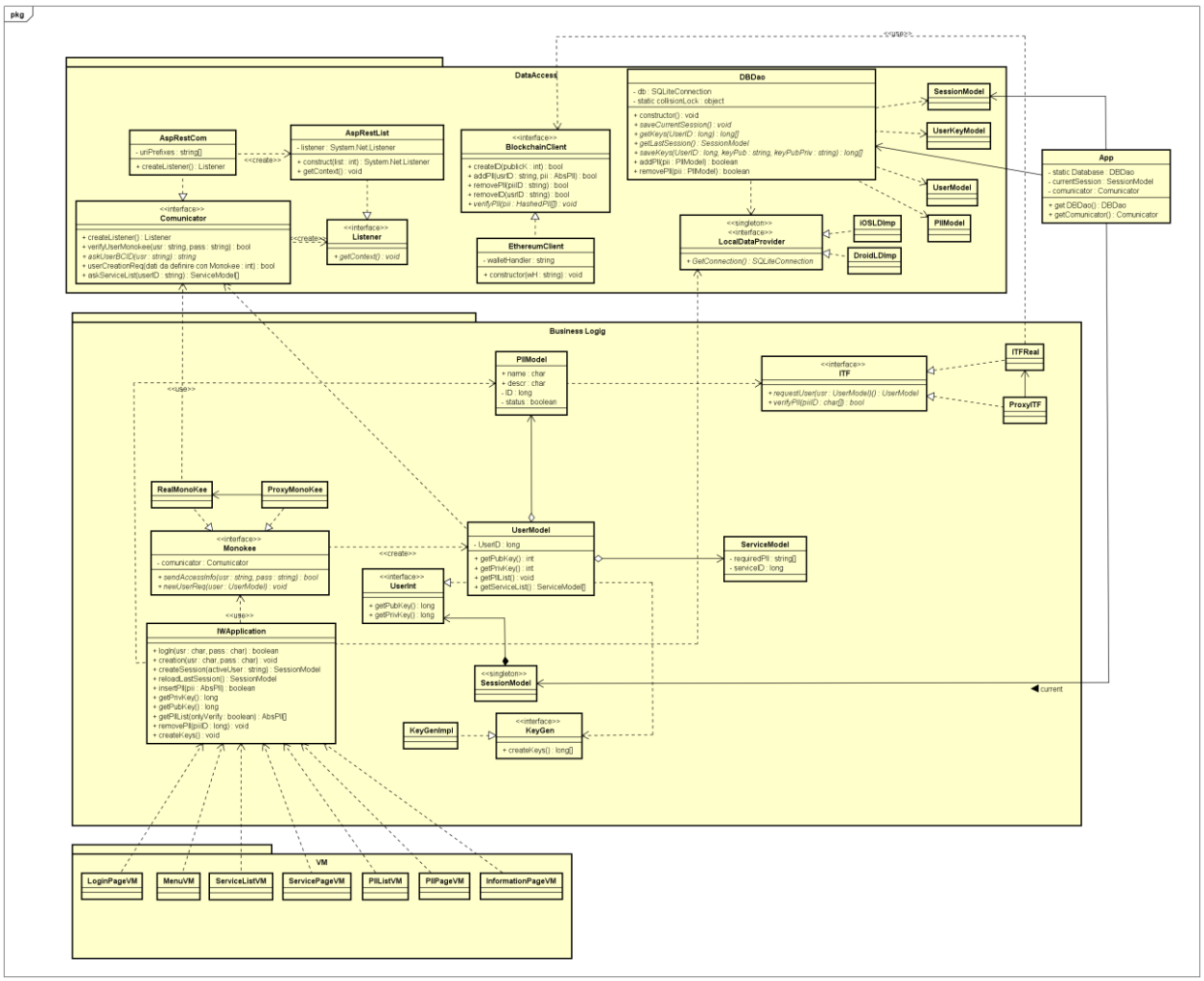
Il documento espone i vari layer presentati in *IW – Architettura di massima* in ogni loro classe e metodo. Viene presentato per ogni layer il loro diagramma UML e per ogni classe una tabella con Descrizione, Parametri, Pseudo codice ed eventuali note.

Riferimenti informativi

- IW – Analisi dei requisiti;
- http://www.html.it/pag/19603/implementare-un-web-service-restful/
- https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api
- https://github.com/antedesk/HelloXamarin/blob/master/HelloXamarin/HelloXamarin/FormViewModel.cs
- https://forums.xamarin.com/discussion/102144/qr-code-generator
- https://developer.xamarin.com/api

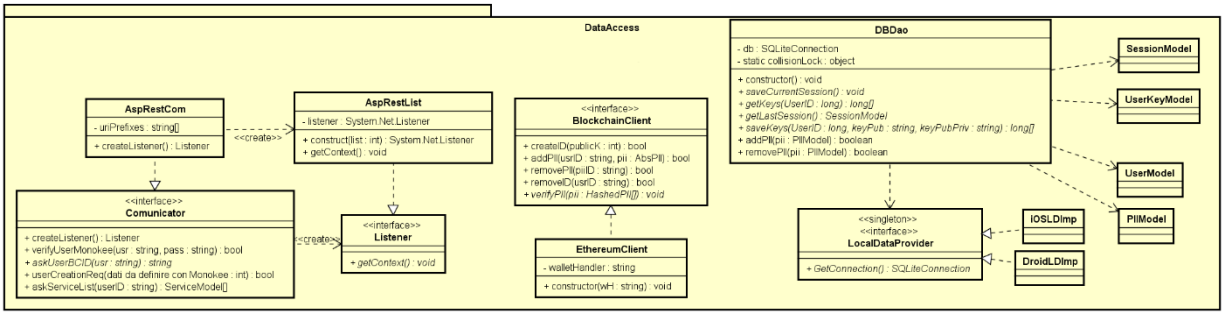
Architettura di dettaglio

Si ricorda come l'architettura scelta in *IW – Architettura di massima* fosse una N-tier con tre strati. Questi sono: DataAccess Layer, BusinessLogic Layer, VMLayer. Nei prossimi capitoli verrà presentato ogni layer in maniera analitica e precisa. Di seguito si mostra il diagramma totale che verrà seguito durante la codifica.



DataAccess Layer

Nel contesto dell'architettura N-tier adottata il BusinessLogic layer è un gruppo di classi che si occupano di interfacciarsi con gli strumenti di persistenza utilizzati dall'applicazione. Questi sono: Monokee (tramite RESTful), ITF e una base di dati locale al dispositivo.



Comunicator (interfaccia)

Questa classe fornisce un'interfaccia per gestire tutte le informazioni attraverso fonti esterne. Deve essere usata per comunicare con Monokee.

Metodi:

Public Listener createListener()

Descrizione	Il metodo ha il compito di restituire un oggetto listener che ascolto le richieste da parte di Monokee.
Parametri	
Pseudo codice	Non presente
Note	

Public bool verifyUserMonokee(usr:string, pass:string)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine di verificare i dati forniti dall'utente. Ritorna true le l'autenticazione ha avuto successo altrimenti false.
Parametri	• usr: string che rappresenta la chiave dell'utente • pass: string che rappresenta la password con cui l'utente usr tenta di effettuare l'accesso. Verificare poi come trasportare la password.
Pseudo codice	Non presente
Note	

Public string askUserBCID(usr:string)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine di restituire una stringa contenente l'ID sull'ITFdell'utente.
Parametri	• usr: string che rappresenta la chiave dell'utente
Pseudo codice	Non presente
Note	

Public bool userCreationRequest(usr: userModel)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine creare un utente all'interno del servizio Monokee. Questo metodo

	invia solo la richiesta e non ha modo di sapere il reale inserimento dell'utente. L'operazione in Monokee non è immediata.
Parametri	• usr: riferimento ad un oggetto UserModel che contiene i dati che l'utente vuole creare.
Pseudo codice	Non presente
Note	Non bisogna dare per scontato che la richiesta riceva esito positivo o che venga eseguita in maniera immediata.

Public ServiceModel[] getServiceList(usrID:string)

Descrizione	Il metodo ha il compito ritornare la lista dei servizi associati all'utente indicato in Monokee.
Parametri	• usrID: stringa che rappresenta la chiave dell'utente in Monokee.
Pseudo codice	Var com = App.getComunicator(); Return serList = await com.askServiceList(this.id);
Note	Questa informazione deve essere richiesta ogni volta, in quanto l'unico gestore di utenti e servizi e Monokee.

AspRestCom (implementa Comunicator)

Campi dati:

uriPrefixes: lista stringhe che rappresentano gli uri a cui il listener ascolta.

Metodi:

Public Listener createListener()

Descrizione	Il metodo ha il compito di restituire un oggetto listener che ascolto le richieste da parte di Monokee.
Parametri	

Pseudo codice	check (uriPrefixes not null);
	check(uriPrefixes not empty)
	listener = HttpListener;
	foreach (uri in uriPrefixes) {
	  listener.add(uri);
	}
	listener.start();
	return new AspRest(listener);

Note	

Public bool verifyUserMonokee(usr:string, pass:string)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine di verificare i dati forniti dall'utente. Ritorna true le l'autenticazione ha avuto successo altrimenti false.
Parametri	• usr: long che rappresenta la chiave dell'utente • pass: string che rappresenta la password con cui l'utente usr tenta di effettuare l'accesso. Verificare poi come trasportare la password.
Pseudo codice	string content =" accessReq"+usr+pass; ASCIIEncoding encoding=new ASCIIEncoding();

	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-IWAccessRequest ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();

Note	http://www.dotnethell.it/tips/SendPOSTHttp.aspx

Public string askUserBCID(usr:string)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine di restituire una stringa contenente l'ID sull'ITF dell'utente.
Parametri	• usr: string che rappresenta la chiave dell'utente

Pseudo codice	string content =”ITF ID request”+usr;
	ASCIIEncoding encoding=new ASCIIEncoding();
	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-ITFIDrequest ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();
	if (response != no pass) return response;
	else return null;

Note	http://www.dotnethell.it/tips/SendPOSTHttp.aspx

Public bool userCreationRequest(usr: userModel)

Descrizione	Il metodo ha il compito di inviare una richiesta a Monokee al fine creare un utente all'interno del servizio Monokee. Questo metodo invia solo la richiesta e non ha modo di sapere il reale inserimento dell'utente. L'operazione in Monokee non è immediata.
Parametri	• usr: riferimento ad un oggetto UserModel che contiene i dati che l'utente vuole creare.

Pseudo codice	string content =”Monokee user creation request”+usr;
	ASCIIEncoding encoding=new ASCIIEncoding();
	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-MONK-creationReq ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();
	if (response != no pass) return response;
	else return null;

Note	Non bisogna dare per scontato che la richiesta riceva esito positivo o che venga eseguita in maniera immediata.

Listener (interfaccia)

È un oggetto con il compito di rimanere in ascolto su determinati uri. È un oggetto non mutabile.

Metodi:

Public void getContext()

Descrizione	Il metodo ha il compito di ritornare appena arriva un messaggio inviato da uno degli uri specificati nella AspRestComp.
Parametri	
Pseudo codice	Non presente
Note	

AspRestList (implementa Listerner)

È un oggetto con il compito di rimanere in ascolto su determinati uri. È un oggetto non mutabile. Questa classe rappresenta un wrapper del listener di System.Net.

Campi dati:

listener: è il listener fornito dal System.Net. Creato da AspRestComp

Metodi:

Public constructor()

Descrizione	Il metodo ha il compito di costruire l'oggetto Listener da un'istanza del listener fornito da System.Net
Parametri	• List: è un'implementazione di System.Net listener.
Pseudo codice	This.listener = List;
Note	

Public void getContext()

Descrizione	Il metodo ha il compito di ritornare appena arriva un messaggio inviato da uno degli uri specificati nella AspRestComp.
Parametri	
Pseudo codice	Listener.getContext();
Note	

BlockchainClient (interfaccia)

Questa interfaccia ha il compito di rappresentare un canale di comunicazione verso gli SmartContract. Deve essere atea rispetto alla tipologia di blockchain usata.

Metodi:

public bool verifyPII(pii: HashedPII[])

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che verifica i dati.
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

public bool createID(publicK : long)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che inserisce un utente nell'ITF.
Parametri	• publicK: è un long che rappresenta la chiave pubblica dell'utente creato. La chiave privata deve rimanere solo in locale nell'IW.

Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

public bool addPII(usrID:string, pii:AbsPII)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che inserisce una nuova PII ad un utente.
Parametri	• usrID: è una stringa che rappresenta la chiave pubblica dell'utente a cui si vuole creare. • Pii: è una lista di oggetti AbsPII che si vuole aggiungere all'utente identificato dalla chiave.
Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

public bool removePII(piiID:string)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che elimina una determinata PII ad un utente.
Parametri	• piiID: è una stringa che rappresenta la chiave della PII che si vuole elimare.
Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

public bool removeID(usrID:string)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che elimina un determinato utente.
Parametri	• usrID: è una stringa che rappresenta la chiave dell'utente che si vuole eliminare.
Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

NethereumClient (implementa BlockchainClient)

Questa classe rappresentare un canale di comunicazione verso gli SmartContract di una rete Ethereum. Fa uso della libreria .NET Nethereum per instaurare la comunicazione.

Campi dati:

walletHandler: è una stringa che rappresenta l'indirizzo del contratto WalletHandler all'interno della blockchain.

Metodi*:*

public constructor(walletHandler: string)

Descrizione	Il metodo ha il compito di costruire l'oggetto Ethereum client.
Parametri	• walletHandler: è una stringa che rappresenta l'indirizzo del contratto WalletHandler.
Pseudo codice	This.walletHandler = walletHandler;

Note	

Public bool verifyPII(pii: HashedPII[])

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che verifica i dati.
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Nethereum.Web3.Web3(); web3. Eth. Transactions.verifyMethod.Call; web3. Eth. Transactions.GetTransactionReceipt; `return result;`
Note	http://nethereum.com/

public bool createID(publicK : long)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che inserisce un utente nell'ITF.
Parametri	• publicK: è un long che rappresenta la chiave pubblica dell'utente creato. La chiave privata deve rimanere solo in locale nell'IW.

Pseudo codice	Nethereum.Web3.Web3();
	trova abi;
	var contract = web3.Eth.GetContract(abi, walletHandler);
	var createFunction = contract.GetFunction("createUser");
	var result = await createFunction.CallAsync<string>(id);

Note	www.nethereum.com

public bool addPII(usrID:string, pii:AbsPII)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che inserisce una nuova PII ad un utente.
Parametri	• usrID: è una stringa che rappresenta la chiave pubblica dell'utente a cui si vuole creare. • Pii: è una lista di oggetti AbsPII che si vuole aggiungere all'utente identificato dalla chiave.
Pseudo codice	Nethereum.Web3.Web3(); Trova abi; Var contract= web3.Eth.GetContract(abi, walletHandler); Var addPIIFunction = contract.GetFunction("addPII"); Var result = await addPIIFunction.CallAsync<PII>(pii);
Note	www.nethereum.com, la chiave deve essere fornita in quanto questa verrà usata anche nel contesto dell'ITF.

public bool removePII(piiID:string)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che elimina una determinata PII ad un utente.
Parametri	• piiID: è una stringa che rappresenta la chiave della PII che si vuole elimare.

Pseudo codice	Nethereum.Web3.Web3();
	trova abi(walletHandler);
	var contract = web3.Eth.GetContract(abi, walletHandler);
	var removeFuncition = contract.GetFunction("removePII");

	var result = await removeFunction.CallAsync<string>(piiID);
Note	www.nethereum.com

public bool removeID(usrID:string)

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che elimina un determinato utente.
Parametri	• usrID: è una stringa che rappresenta la chiave dell'utente che si vuole eliminare.

Pseudo codice	Nethereum.Web3.Web3();
	trova abi(walletHandler);
	var contract = web3.Eth.GetContract(abi, walletHandler);
	var removeFuncition = contract.GetFunction("removeID");
	var result = await removeFunction.CallAsync<string>(usrID);

Note	www.nethereum.com

DBDao

Questa classe ha il compito di fare da tramite per gli accessi al database. Rappresenta quindi il data access object relativo al database. Per effettuare la connessione viene usata la classe LocalDataProvider.

Campi dati:

database: è un oggetto di tipo SQLiteConnection che rappresenta la connessione con il database

static collisionLock: è un oggetto di qualsiasi tipo con lo scopo di fornire un lock all'oggetto per non permettere usi concorrenti della risorsa.

Metodi:

public constructor()

Descrizione	Il metodo ha il compito costruire l'oggetto DBDao
Parametri	
Pseudo codice	database = DependencyService.Get<IRecipesDatabaseConnection>().GetConnection(); database.CreateTable<PIIModel>(); database.CreateTable<SessionModel>(); database.CreateTable<UserModel>();

Note	Questo metodo deve essere chiamato nella classe App del progetto nel seguente modo.
	`public static DBDao Database`
	`{`
	`  get`
	`  {`
	`  if (database == null)`
	`  {`
	`  database = new DBDao();`
	`  }`
	`  return database;`
	`  }`
	`}`
	Questo serve per creare l'unica istanza della base di dati all'avvio dell'applicazione.

public saveCurrentSession()

Descrizione	Il metodo ha il compito salvare all'interno del database la sessione corrente
Parametri	

Pseudo codice	`var sessionModel = ((IW)Application).currentSession;`
	lock (collisionLock)
	{
	if (`sessionModel`.Id != 0)
	{
	database.Update(`sessionModel`);
	return `sessionModel`.Id;
	}
	else
	{
	return database.Insert(`sessionModel`);
	}
	}

Note	Il controllo è utile nel contesto in cui id è auto incrementale, infatti in un model creato non dal database questo viene settato a 0 di default. Anche usando gli ID provenienti dalla blockchain la cosa rimane vera in quanto non esisterebbe l'indirizzo 0.

public sessionModel getLastSession()

Descrizione	Il metodo ha il compito ritornare l'ultima sessione avviata
Parametri	

Pseudo codice	Var last id = codice che trova il last id;
	lock (collisionLock)
	{
	return database.Table<sessionModel>().FirstOrDefault(x => x.Id == id);
	}

Note	

public UserKeyModel getKeys(userID:long)

Descrizione	Il metodo ha il compito ritornare la coppia di chiavi generate per l'utente specificato nel primo parametro.
Parametri	• userID: è un long che rappresenta la chiave dell'ID presente nella base di dati.

Pseudo codice	Var last id = codice che trova il last id;
	lock (collisionLock)
	{
	return database.Table<sessionModel>().FirstOrDefault(x => x.Id == userID);
	}

Note	

Public void SaveUserKeys(usrID:string, keyPub:string, keyPriv:string)

Descrizione	Il metodo ha il compito salvare all'interno del database la sessione corrente

Parametri	• usrID: stringa che rappresenta l'utente a cui fanno riferimento le chiavi • keyPub: stringa che contiene la chiave pubblica che si vuole attribuire • keyPriv: stringa che contiene la chiave privata che si vuole attribuire

Pseudo codice	`var keys = new UserKeysModel(usrID,keyPub,keyPriv);`
	lock (collisionLock)
	{
	if (keys.userId non presente)
	{
	database.Update(keys);
	return keys.Id;
	}
	else
	{
	return database.Insert(keys);
	}
	}

Note	Il controllo è utile nel contesto in cui id è auto incrementale, infatti in un model creato non dal database questo viene settato a 0 di default. Anche usando gli ID provenienti dalla blockchain la cosa rimane vera in quanto non esisterebbe l'indirizzo 0.

Public long addPII(pii: PIIModel)

Descrizione	Il metodo ha il compito salvare una PII all'interno del database. La chiave ritornata è quella decisa dall'auto incremento del database.
Parametri	• pii: è un riferimento ad un oggetto PIIModel che contiene le informazioni che si vogliono inserire all'interno della base di dati.

Pseudo codice	lock (collisionLock)
	{
	if (pii.ID non presente)
	{
	database.Update(pii);
	return pii.Id;
	}
	else
	{
	database.Insert(pii);
	return pii.Id;
	}
	}

Note	Questa chiave identifica la PII e deve essere comunicata all'ITF.

Public long removePII(piiID: string)

Descrizione	Il metodo ha il compito rimuovere una PII all'interno del database.
Parametri	• piiID: è un riferimento ad una stringa che identifica la chiave della PII.

Pseudo codice	lock (collisionLock)
	{
	return database.Delete<PIIModel>(piiID);
	}

Note	Si ricorda che la chiave è usata anche nel contesto dell'ITF.

LocalDataProvider (interfaccia, singleton)

Questa interfaccia una connessione con un database locale SQLite, Questa interfaccia deve poi essere implementata a seconda della piattaforma.

Metodi:

public SQLiteConnection GetConnection()

Descrizione	Il metodo ha il compito di stabilire la connessione con il database SQLite residente nel dispositivo. Questo metodo restituisce una connessione al database.
Parametri	
Pseudo codice	
Note	https://developer.xamarin.com/guides/android/data-and-cloud-services/data-access/part-3-using-sqlite-orm/

DroidLDImpl (implementa LocalDataProvider)

Questa classe fornisce un'implementazione di LocalDataProvider per il sistema Android.

Metodi:

public SQLiteConnection GetConnection()

Descrizione	Il metodo ha il compito di stabilire la connessione con il database SQLite residente nel dispositivo. Questo metodo restituisce una connessione al database.
Parametri	

Pseudo codice	Var sqliteFilename = "recipesDB.db3";
	String docuPath = System.Enviroment.GetFolderPath(`System.Environment.SpecialFolder.Personal`);
	`Var path = Path.Combine(documetnsPath, sqliteFilename);`
	`Var conn = new SQLite.SQLiteConnection(path);`
	`Return conn;`

Note	https://developer.xamarin.com/guides/android/data-and-cloud-services/data-access/part-3-using-sqlite-orm/

iOSLDImpl (implementa LocalDataProvider)

Questa classe fornisce un'implementazione di LocalDataProvider per il sistema Android.

Metodi:

public SQLiteConnection GetConnection()

Descrizione	Il metodo ha il compito di stabilire la connessione con il database SQLite residente nel dispositivo. Questo metodo restituisce una connessione al database.
Parametri	
Pseudo codice	Var sqliteFilename = "RecipesDB.db3";

	String documentsPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal); String libraryPath = Path.Combine(documets,"..","Library"); Var path = path.Combine(libraryPath, sqliteFilename); Var conn = new SQLite.SQLiteConnection(path); Return conn;
Note	https://developer.xamarin.com/guides/android/data-and-cloud-services/data-access/part-3-using-sqlite-orm/

PIIModel (implementa INotifyPropertyChanged)

Questa classe rappresenta un elemento della tabella nella base di dati delle PII inserite nel dispositivo.

[Table("PIIs")]

Campi dati:

name: rappresenta il nome della PII [NotNull]

desc: rappresenta una string con la descrizione della PII.

ID: rappresenta la chiave della PII nella base di dati e nell'ITF. [PrimaryKey, AutoIncrement] La chiave viene decisa dal database e sarà la stessa usata anche nel ITF.

status: è un bool che se a true indica che la PII è stata verificata nell'ITF, se a false indica che la PII o non è stata validata.

Metodi:

Ogni attributo deve avere un getter e un setter, questi andranno implementati seguendo le funzionalità che offre C#. Ogni setter deve chiamare PropertyChanged al fine di garantire un corretto data binding.

private void PropertyChanged (propertyName)

Descrizione	Il metodo ha il compito di effettuare il databinding con gli oggetti che modellano la vista e il database.
Parametri	propertyName: il nome della variabile a cui è stato effettuato il cambiamento.
Pseudo codice	this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
Note	Questo metodo deve essere chiamato da ogni set presente nel seguente modo OnPropertyChanged(nameof(nome));

SessionModel (implementa INotifyPropertyChanged)

Questa classe rappresenta un elemento della tabella nella base di dati delle Sessions inserite nel dispositivo.

[Table("Sessions")]

Campi dati:

activeUser: è un long che rappresenta la chiave all'interno del database dell'ID che ha effettuato il login nella sessione. [Foreignkey User.id]

ID: rappresenta la chiave della sessione nella base di dati locale. [PrimaryKey, AutoIncrement]

In questa prima fase non sono stati definiti altri attributi, ma potrebbero essere inseriti informazioni di log quali timestamp di accesso e di log out.

Metodi:

Ogni attributo deve avere un getter e un setter, questi andranno implementati seguendo le funzionalità che offre C#. Ogni setter deve chiamare PropertyChanged al fine di garantire un corretto data binding.

private void PropertyChanged (propertyName)

Descrizione	Il metodo ha il compito di effettuare il databinding con gli oggetti che modellano la vista e il database.
Parametri	propertyName: il nome della variabile a cui è stato effettuato il cambiamento.
Pseudo codice	this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
Note	Questo metodo deve essere chiamato da ogni set presente nel seguente modo OnPropertyChanged(nameof(nome));

UserModel (implementa INotifyPropertyChanged, implementa UserInt)

Questa classe rappresenta un elemento della tabella nella base di dati dell'utente inserite nel dispositivo.

[Table("Users")]

Campi dati:

activeUser: è un long che rappresenta la chiave all'interno del database dell'ID che ha effettuato il login nella sessione. [Foreignkey User.id]

PIIList: è una lista di chiavi delle PII presenti nel DB. Queste devo essere rappresentate in una stringa unica in modo da rendere più agevole l'inserimento nella base di dati. Il getter di questa proprietà deve essere restituita come array di chiavi e non come stringa unica.

ID: rappresenta la chiave dell'user nella base di dati locale, eventualmente da fare coincidere con la chiave dell'utente nell'ITF. [PrimaryKey, AutoIncrement (in caso le chiavi non coincidano)]

Metodi:

Ogni attributo deve avere un getter e un setter, questi andranno implementati seguendo le funzionalità che offre C#. Ogni setter deve chiamare PropertyChanged al fine di garantire un corretto data binding.

private void PropertyChanged (propertyName)

Descrizione	Il metodo ha il compito di effettuare il databinding con gli oggetti che modellano la vista e il database.
Parametri	propertyName: il nome della variabile a cui è stato effettuato il cambiamento.
Pseudo codice	this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
Note	Questo metodo deve essere chiamato da ogni set presente nel seguente modo OnPropertyChanged(nameof(nome));

public void getPrivKey()

Descrizione	Il metodo ha il compito ritornare la chiave privata presenta nella base di dati se presente. Se non presente genera la coppia e la ritorna.
Parametri	

Pseudo codice	Var dbDao = App.getDBDao();
	lock (collisionLock)
	{
	Keys = dbDao.getKeys(id);
	}
	If Keys == null {
	Var gen =DependencyService.Get<KeyGen>();
	Long[] keys = gen.createKeys();
	dbDao.saveKeys(id, keys[0],keys[1]);
	}
	return Keys[0];

Note	

public void getPubKey()

Descrizione	Il metodo ha il compito ritornare la chiave privata presenta nella base di dati se presente. Se non presente genera la coppia e la ritorna.
Parametri	

Pseudo codice	Var dbDao = App.getDBDao();
	lock (collisionLock)
	{
	Keys = dbDao.getKeys(id);
	}
	If Keys == null {
	Var gen =DependencyService.Get<KeyGen>();
	Long[] keys = gen.createKeys();
	dbDao.saveKeys(id, keys[0],keys[1]);
	}
	return Keys[1];

Note	

Public ServiceModel[] getServiceList()

Descrizione	Il metodo ha il compito ritornare la lista dei servizi associati all'utente impostato come active user in Monokee.
Parametri	
Pseudo codice	Var com = App.getComunicator(); var id = App.currentSession.activeUser.id; Return serList = await com.askServiceList(this.id);
Note	Questa informazione deve essere richiesta ogni volta, in quanto l'unico gestore di utenti e servizi e Monokee.

UserKeyModel (implementa INotifyPropertyChanged)

Questa classe rappresenta un elemento della tabella nella base di dati delle Sessions inserite nel dispositivo.

[Table("UserKeys")]

Campi dati:

user: è un long che rappresenta la chiave dell'utente all'interno del database di cui fanno riferimento le informazioni. [Foreignkey User.id, PrimaryKey]

KeyPriv: rappresenta la chiave pubblica. [NotNull]

KeyPriv: rappresenta la chiave privata. [NotNull]

Metodi:

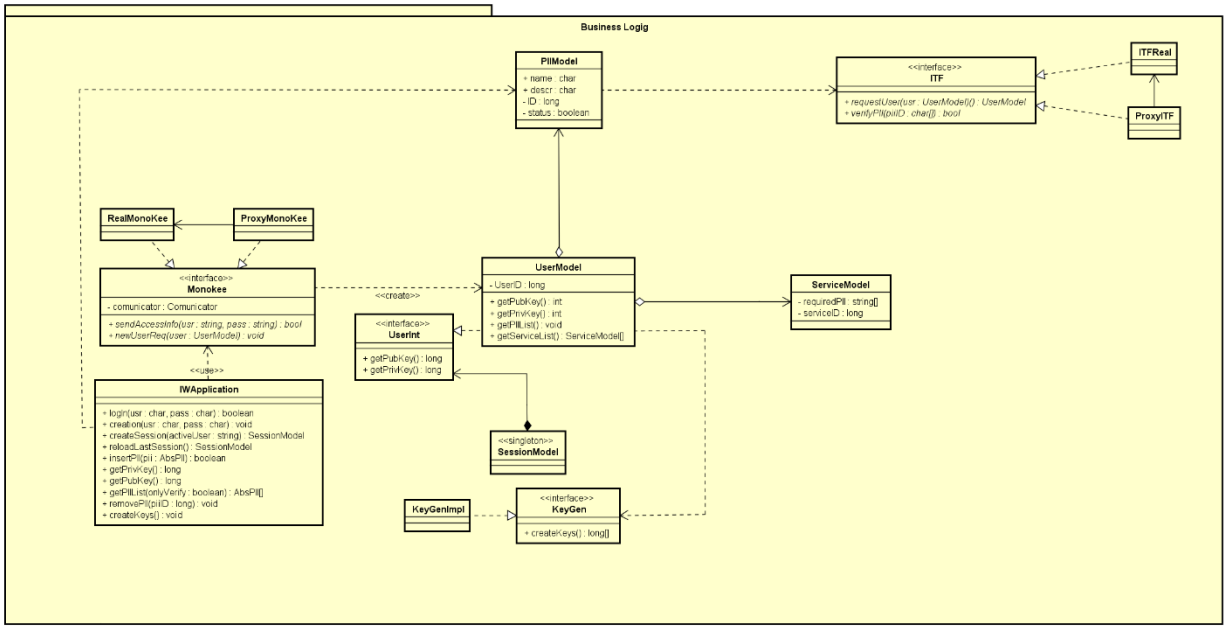
Ogni attributo deve avere un getter e un setter, questi andranno implementati seguendo le funzionalità che offre C#. Ogni setter deve chiamare PropertyChanged al fine di garantire un corretto data binding.

private void PropertyChanged (propertyName)

Descrizione	Il metodo ha il compito di effettuare il databinding con gli oggetti che modellano la vista e il database.
Parametri	propertyName: il nome della variabile a cui è stato effettuato il cambiamento.
Pseudo codice	this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
Note	Questo metodo deve essere chiamato da ogni set presente nel seguente modo OnPropertyChanged(nameof(nome));

BusinessLogic layer

Nel contesto dell'architettura N-tier adottata il BusinessLogic layer è un gruppo di classi che si occupano di effettuare e di mantenere tutte le regole definite dai documenti IW - Analisi di massima, IW – Studio di fattibilità.



Monokee (interfaccia)

Questa interfaccia rappresenta l'entità Monokee, con le classi RealMonokee e MonokeeProxy partecipa ad un'applicazione di un proxy pattern.

Metodi:

Public async bool sendAccessInfo(userID: string, pass:string)

Descrizione	Il metodo ha il compito di restituire true in caso i parametri passati corrispondano ad un username e una password di un utente presente nel servizio Monokee. False altrimenti. La chiamata è asincrona.
Parametri	• userID: stringa che rappresenta la chiave dell'utente • pass: stringa che rappresenta la password dell'utente in chiaro.
Pseudo codice	Non presente
Note	

Public void newUserRequest(user:UserModel)

Descrizione	Il metodo ha il compito di creare una richiesta di creazione utente in Monokee. Il metodo invia solo la richiesta, non ha modo di sapere se l'utente venga creato o meno.
Parametri	• user: è un riferimento ad un oggetto userModel che contiene i dati con cui si vuole inviare la richiesta di creazione l'utente.
Pseudo codice	Non presente
Note	

RealMonokee (implementa Monokee)

Rappresenta una reale istanza del servizio Monokee. Con Monokee e MonokeeProxy rappresenta un'applicazione del pattern Proxy.

Campi dati:

comunicator: è un riferimento di un oggetto che implementa l'interfaccia Comunicator.

Metodi:

Public async bool sendAccessInfo (userID: string, pass:string)

Descrizione	Il metodo ha il compito di restituire true in caso i parametri passati corrispondano ad un username e una password di un utente presente nel servizio Monokee. False altrimenti.
Parametri	• userID: long che rappresenta la chiave dell'utente • serviceID: long che rappresenta la chiave del servizio richiesto
Pseudo codice	Return Async Communicator.verifyUserMonokee(userID,serviceID);
Note	

Public void newUserRequest(user:UserModel)

Descrizione	Il metodo ha il compito di creare una richiesta di creazione utente in Monokee. Il metodo invia solo la richiesta, non ha modo di sapere se l'utente venga creato o meno.
Parametri	• user: è un riferimento ad un oggetto userModel che contiene i dati con cui si vuole inviare la richiesta di creazione l'utente.
Pseudo codice	Communicator.userCreationReq(user:UserModel);
Note	

MonokeeProxy (implementa Monokee)

Rappresenta un proxy remoto del servizio Monokee. Applica una politica di acquisizione pigra. Con Monokee e RealMonokee rappresenta un'applicazione del pattern Proxy.

Campi dati:

realMonokee: è un riferimento di un oggetto RealMonokee.

Metodi:

Public async bool sendAccessInfo (userID: string, pass:string)

Descrizione	Il metodo ha il compito di restituire true in caso i parametri passati corrispondano ad un username e una password di un utente presente nel servizio Monokee. False altrimenti.
Parametri	• userID: long che rappresenta la chiave dell'utente • serviceID: long che rappresenta la chiave del servizio richiesto
Pseudo codice	applyPolicy1; applyPolicy2; …. applyPolicyN; realMonokee.sendAccessInfo(userID, pass);
Note	

Public void newUserRequest(user:UserModel)

Descrizione	Il metodo ha il compito di creare una richiesta di creazione utente in Monokee. Il metodo invia solo la richiesta, non ha modo di sapere se l'utente venga creato o meno.
Parametri	• user: è un riferimento ad un oggetto userModel che contiene i dati con cui si vuole inviare la richiesta di creazione l'utente.
Pseudo codice	applyPolicy1; applyPolicy2; …. applyPolicyN; realMonokee.newUserRequest(user);
Note	

UserInt (interfaccia)

Questa interfaccia definisce le caratteristiche minime che deve avere un utente generico dell'applicazione. Queste proprietà sono le chiavi private e pubbliche. UserModel implementa questa interfaccia. Questo ADT è stato pensato in un'ottica futura in cui ci potrebbero essere utenti non di Monokee.

Metodi:

public void getPrivKey()

Descrizione	Il metodo ha il compito ritornare la chiave privata presenta nella base di dati se presente. Se non presente genera la coppia e la ritorna.
Parametri	
Pseudo codice	Non presente
Note	

public void getPubKey()

Descrizione	Il metodo ha il compito ritornare la chiave privata presenta nella base di dati se presente. Se non presente genera la coppia e la ritorna.
Parametri	
Pseudo codice	Non presente.
Note	

ServiceModel (implementa INotifyPropertyChanged)

Questa classe rappresenta un servizio a cui l'utente può avere accesso. È definito dalla un codice e da una lista di PII richieste per effettuare l'accesso. Questo oggetto dovrà essere sempre generato da Monokee e mai salvato in locale. Questo al fine di evitare dati discordanti.

Campi dati:

serviceID: è una stringa che rappresenta una chiave del servizio all'interno di Monokee.

requiredPII[]: è un array di stringhe che rappresentano gli ID delle PII necessarie per effettuare il login al servizio identificato da serviceID

Metodi:

Ogni attributo deve avere un getter e un setter, questi andranno implementati seguendo le funzionalità che offre C#. Ogni setter deve chiamare PropertyChanged al fine di garantire un corretto data binding.

private void PropertyChanged (propertyName)

Descrizione	Il metodo ha il compito di effettuare il databinding con gli oggetti che modellano la vista e il database.
Parametri	propertyName: il nome della variabile a cui è stato effettuato il cambiamento.
Pseudo codice	this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
Note	Questo metodo deve essere chiamato da ogni set presente nel seguente modo OnPropertyChanged(nameof(nome));

Keygen (interfaccia)

Questa interfaccia rappresenta un Strategy pattern per nascondere l'implementazione della reale libreria che effettua la generazione delle chiavi. Ha anche il compito di criptare e decriptare i dati.

Metodi:

public long [] createKeys()

Descrizione	Il metodo ha il compito di creare una coppia di chiavi pubbliche e private e quindi di restituirle. Ritorna un array di string di due elementi. Il primo elemento è la chiave pubblica, il secondo la privata.
Parametri	
Pseudo codice	Non presente.
Note	

public static byte[] Encrypt(string publicKey, string data)

Descrizione	Il metodo ha il compito di firmare i dati forniti con la chiave pubblica fornita.
Parametri	• publicKey: stringa che rappresenta la chiave pubblica • Data: stringa che rappresenta i dati che si vogliono firmare
Pseudo codice	Non presente.
Note	

public static byte[] Encrypt(string publicKey, string data)

Descrizione	Il metodo ha il compito di firmare i dati forniti con la chiave pubblica fornita.
Parametri	• publicKey: stringa che rappresenta la chiave pubblica • Data: stringa che rappresenta i dati che si vogliono firmare
Pseudo codice	Non presente.
Note	

public static string Decrypt(string privateKey, byte[] encryptedBytes)

Descrizione	Il metodo ha il compito di decriptare i dati forniti con la chiave privata fornita.
Parametri	• privateKey: stringa che rappresenta la chiave privata • encryptedBytes: array di byte che rappresentano i dati che si vogliono decriptare.
Pseudo codice	Non presente
Note	

KeygenImpl (implementa KeyGen)

Questa interfaccia rappresenta un template pattern per nascondere l'implementazione della reale libreria che effettua la generazione delle chiavi.

Metodi:

public static string Decrypt(string privateKey, byte[] encryptedBytes)

Descrizione	Il metodo ha il compito di creare una coppia di chiavi pubbliche e private e quindi di restituirle. Ritorna un array di string di due elementi. Il primo elemento è la chiave pubblica, il secondo la privata.
Parametri	

Pseudo codice	CspParameters cspParams = new CspParameters { ProviderType = 1 };
	
	RSACryptoServiceProvider rsaProvider = new RSACryptoServiceProvider(1024, cspParams);
	
	string publicKey = Convert.ToBase64String(rsaProvider.ExportCspBlob(false));
	string privateKey = Convert.ToBase64String(rsaProvider.ExportCspBlob(true));
	
	return new array = {privateKey, publicKey};

Note	https://stackoverflow.com/questions/18850030/aes-256-encryption-public-and-private-key-how-can-i-generate-and-use-it-net

public static byte[] Encrypt(string publicKey, string data)

Descrizione	Il metodo ha il compito di firmare i dati forniti con la chiave pubblica fornita.
Parametri	• publicKey: stringa che rappresenta la chiave pubblica • Data: stringa che rappresenta i dati che si vogliono firmare

Pseudo codice	CspParameters cspParams = new CspParameters { ProviderType = 1 };
	RSACryptoServiceProvider rsaProvider = new RSACryptoServiceProvider(cspParams);
	
	rsaProvider.ImportCspBlob(Convert.FromBase64String(publicKey));
	
	byte[] plainBytes = Encoding.UTF8.GetBytes(data);
	byte[] encryptedBytes = rsaProvider.Encrypt(plainBytes, false);
	
	return encryptedBytes;

Note	https://stackoverflow.com/questions/18850030/aes-256-encryption-public-and-private-key-how-can-i-generate-and-use-it-net

public static string Decrypt(string privateKey, byte[] encryptedBytes)

Descrizione	Il metodo ha il compito di decriptare i dati forniti con la chiave privata fornita.
Parametri	• privateKey: stringa che rappresenta la chiave privata

	• encryptedBytes: array di byte che rappresentano i dati che si vogliono decriptare.

Pseudo codice	CspParameters cspParams = new CspParameters { ProviderType = 1 };
	RSACryptoServiceProvider rsaProvider = new RSACryptoServiceProvider(cspParams);
	
	rsaProvider.ImportCspBlob(Convert.FromBase64String(privateKey));
	
	byte[] plainBytes = rsaProvider.Decrypt(encryptedBytes, false);
	
	string plainText = Encoding.UTF8.GetString(plainBytes, 0, plainBytes.Length);
	
	return plainText;

Note	https://stackoverflow.com/questions/18850030/aes-256-encryption-public-and-private-key-how-can-i-generate-and-use-it-net

ITF (interfaccia)

Questa interfaccia rappresenta il componente ITF. Con RealITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Metodi:

public verifyPII(pii: PIIModel[]):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII. Il metodo si occupa della creazione dell'Hash.
Parametri	• pii: è una lista di oggetti PIIModel da verificare nell'ITF.
Pseudo codice	Non presente.
Note	

public requestUser(usr: userModel[]):bool

Descrizione	Il metodo ha il compito creare un utente all'interno dell'ITF.
Parametri	• usr: rappresenta l'utente che si vuole immettere nell'ITF
Pseudo codice	Non presente.
Note	La chiave pubblica rappresenta l'ID dell'utente.

RealITF (implementa ITF)

Questa interfaccia rappresenta il reale componente ITF. Si occupa di instaurare la comunicazione tramite l'uso di un BlockchainClient. Con ITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Campi dati:

tutti gli oggetti necessari vengono ottenuti tramite l'uso del DependencyService.

Metodi:

public verifyPII(pii: PIIModel []):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Var blockClient = DependencyService.Get<BlockchainClient>();

	HashAlgorithm algorithm = SHA256.Create(); Var hasedDesc = algorithm.ComputeHash(Encoding.UTF8.GetBytes(pii.desc)); Var hashedName = algorithm.ComputeHash(Encoding.UTF8.GetBytes(pii.name)); Return blockClient.createID(piiModel.id,hashedName, hashedDesc);
Note	

public bool requestUser(usr: userModel[])

Descrizione	Il metodo ha il compito creare un utente all'interno dell'ITF.
Parametri	• usr: rappresenta l'utente che si vuole immettere nell'ITF
Pseudo codice	Var blockClient = DependencyService.Get<BlockchainClient>(); Return blockClient.createID(usr.getPublicKey());
Note	La chiave pubblica rappresenta l'ID dell'utente.

ITFProxy (implementa ITF)

Questa interfaccia rappresenta un remote proxy del componente ITF. Si occupa applicare una serie di politiche. Con ITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Campi dati:

realITF: è un riferimento ad un oggetto RealITF.

Metodi:

public verifyPII(pii: PIIModel []):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Return realITF.verifyPII(pii);
Note	

public bool requestUser(usr: userModel[])

Descrizione	Il metodo ha il compito creare un utente all'interno dell'ITF.
Parametri	• usr: rappresenta l'utente che si vuole immettere nell'ITF
Pseudo codice	realITF. requestUser(usr: userModel[]);
Note	La chiave pubblica rappresenta l'ID dell'utente.

IWApplication

Questa classe rappresenta un facade che espone tutte le funzioni che può compiere un utente attraverso le varie viste.

Campi dati:

monokee: è un riferimento ad un oggetto che implementa l'interfaccia Monokee.

database: è un riferimento ad un oggetto DBDao ottenuto tramite l'uso di DependecyService.

Metodi:

public bool logIn(usr: string, pass:string)

Descrizione	Il metodo ha il compito di verificare che le credenziali fornite dall'utente corrispondano anche nel servizio Monokee.

Parametri	• usr: rappresenta la chiave dell'utente • Pass: è una stringa che rappresenta la password dell'utente
Pseudo codice	Var b = Monokee.sendAccessInfo(usr, pass); Return b;
Note	

public void creation(usr: string, pass:string)

Descrizione	Il metodo ha il compito di inviare una richiesta di creazione utente di un utente. Questo metodo invia solo una richiesta, non a modo di sapere se l'utente verrà effettivamente creato.
Parametri	• usr: rappresenta la chiave dell'utente • Pass: è una stringa che rappresenta la password dell'utente
Pseudo codice	Var b = Monokee.newUserRequest(usr, pass); Return ;
Note	

public void createSession(activeUser: string)

Descrizione	Il metodo ha il compito creare la sessione e di inserirla nel database come lastSession.
Parametri	• usr: rappresenta la chiave dell'utente • Pass: è una stringa che rappresenta la password dell'utente
Pseudo codice	currentSession = new SessionModel(activeUser.id); App.currentSession = currentSession; Database.saveCurrentSession();
Note	La current session è mantenuta nell'oggetto App.

public void reloadLastSession()

Descrizione	Il metodo ha il compito ricaricare l'ultima sessione dal database e ritornala.
Parametri	• usr: rappresenta la chiave dell'utente • Pass: è una stringa che rappresenta la password dell'utente
Pseudo codice	Return database.getLastSession();
Note	

public void insertPII(pii:PIIModel)

Descrizione	Il metodo ha il compito inserire la pii all'utente corrente e anche all'interno dell'ITF.
Parametri	• pii: rappresenta la PII che si vuole inserire all'utente corrente
Pseudo codice	Database.addPII(pii,status = false) BlockchainClient.addPII(App.currentSession.activeUser, pii);
Note	

public void getPrivKey()

Descrizione	Il metodo ha il compito ritornare la chiave privata dell'utente che indicato come active user nella sessione corrente.
Parametri	
Pseudo codice	Return BlockchainClient.addPII(App.currentSession.activeUser.getPrivKey();
Note	

public void getPubKey()

Descrizione	Il metodo ha il compito ritornare la chiave Pubblica dell'utente che indicato come active user nella sessione corrente.
Parametri	
Pseudo codice	Return BlockchainClient.addPII(App.currentSession.activeUser.getPubKey();
Note	

public void getPIIList()

Descrizione	Il metodo ha il compito ritornare la lista di PII dell'utente che è indicato come active user nella sessione corrente.
Parametri	
Pseudo codice	Return BlockchainClient.addPII(App.currentSession.activeUser.getPIIList();
Note	

public void removePII(pii:string)

Descrizione	Il metodo ha il compito inserire la pii all'utente corrente e anche all'interno dell'ITF.
Parametri	• pii: rappresenta la chiave dellla PII che si vuole inserire all'utente corrente
Pseudo codice	Database.removePII(pii,status = false) BlockchainClient.removePII(pii);
Note	

public Tuple<string,string> createKeys()

Descrizione	Il metodo ha il compito di creare all'utente corrente un paio di chiavi asincrone.
Parametri	• pii: rappresenta la chiave dellla PII che si vuole inserire all'utente corrente
Pseudo codice	Var a = BlockchainClient.addPII(App.currentSession.activeUser.getPriv(); Var b = BlockchainClient.addPII(App.currentSession.activeUser.getPubl(); Return new Tuple<string,string>(a,b);
Note	L'algoritmo usato è SHA256.

IQRGenerator (interfaccia)

Questa interfaccia rappresenta uno strategy Pattern per accomunare le varie implementazioni di algoritmi che generano una lista di byte che rappresentano immagini di codici QR.

Metodi:

public byte[] generateQR(value: string)

Descrizione	Il metodo ha il compito di creare un'array di byte che rappresentano un'immagine del codice QR contenente la stringa definita in value.
Parametri	• value: è una stringa che rappresenta il testo che si vuole inserire dentro il QR.
Pseudo codice	Return reallTF.verifyPII(pii);
Note	

IQRGenerator (implementa IQRgenerator)

Questa classe rappresenta un'implementazione dell'algoritmo di generazione QR per Android.

Metodi:

public byte[] generateQR(value: string)

Descrizione	Il metodo ha il compito di creare un'array di byte che rappresentano un'immagine del codice QR contenente la stringa definita in value.
Parametri	• value: è una stringa che rappresenta il testo che si vuole inserire dentro il QR.

Pseudo codice	`var writer = new BarcodeWriter`
	`{`
	`Format = BarcodeFormat.QR_CODE,`
	`Options = new EncodingOptions`
	`{`
	`Height = 1600,`
	`Width = 1600`
	`}`
	`};`
	`Android.Graphics.Bitmap bm = writer.Write(value);`
	`MemoryStream stream = new MemoryStream();`
	`bm.Compress(Bitmap.CompressFormat.Jpeg, 100, stream);`
	`var arr = stream.ToArray();`
	`stream.Close();`
	`return arr;`

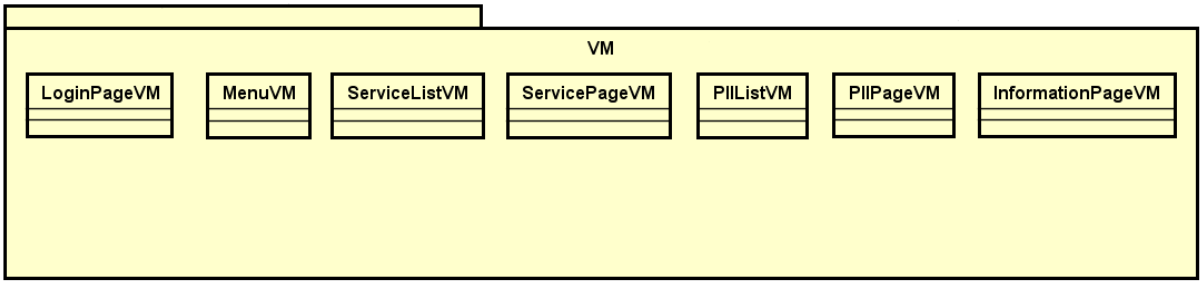
Note	

VM Layer

Questo layer contiene i vari controllori che gestiscono le viste. Si è previsto di creare un controller per ogni pagina dell'applicazione. L'applicazione utilizza il pattern MVVM, quindi, i controlli sono delle ViewModel (VM) che contengono i dati e le operazioni. Tra i dati e la vista sono presenti dei binding da realizzare utilizzando gli strumenti forniti da Xamarin. Le VM operano le loro azioni tramite l'utilizzo della classe IWFacade.

Esiste un controllore per ogni vista prevista dall'applicazione. Le seguenti sono:

- LoginPageVM
- MenuVM
- ServiceListVM
- ServicePageVM
- PIIListVM
- PIIPage
- AddPIIPageVM



Tutte queste classi devono estendere da INotifyPropertyChanged.

LoginPageVM (implementa INotifyPropertyChanged)

È il controllore della pagina di login.

Campi dati:

DisplayInvalidLoginPrompt: è un rifermento ad un oggetto Action (Xamarin) che definisce l'azione da intaprendere in caso di credenziali errate. Questo oggetto è definito nel code-behind della vista e deve essere richiamato tramite una delegate.

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

email: è una stringa in databing con la form presente nella vista.

password: è una stringa in databing con la form presente nella vista.

SubmitCommand: e un riferimento ad un oggeto ICommand(Xamarin). Deve avere setter e getter pubblici.

I campi dati email, password devono avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnSubmit()

Descrizione	Il metodo ha il compito definire le procedure da intraprendere la verifica di email e password.
Parametri	

Pseudo codice	Var result = App.IWFacade.LogIn(email, password);
	If result {
	`Navigation.PushAsync (new ServiceListPage());`
	}
	Else DisplayInvalidLoginPrompt(); //o un eventuale displayAllert

Note	https://www.c-sharpcorner.com/article/xamarin-forms-create-a-login-page-mvvm/

public void OnClickInfo()

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in InfoPage.
Parametri	
Pseudo codice	`Navigation.PushAsync (new InfoPage());`
Note	https://www.c-sharpcorner.com/article/xamarin-forms-create-a-login-page-mvvm/

MenuVM (implementa INotifyPropertyChanged)

È il controllore del menu visualizzato nelle MasterDetaildePage

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

Il campo dati email deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnClickPIIList()

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in PIIListPage.
Parametri	
Pseudo codice	`Navigation.PushAsync (new PIIListPage());`
Note	

public void OnClickKeys()

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in KeysPage.
Parametri	
Pseudo codice	`Navigation.PushAsync (new KeysPage());`
Note	

public void OnClickInfoPage()

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in InfoPage.

Parametri	
Pseudo codice	`Navigation.PushAsync (new InfoPage());`
Note	

KeysPageVM (implementa INotifyPropertyChanged)

È il controllore della pagina che mostra le chiavi pubbliche e private.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

keyPriv: è una stringa in databing, da visualizzare per mostrare la chiave privata dell'active user attuale.

keyPub: è una stringa in databing, da visualizzare per mostrare la chiave pubblica dell'active user attuale.

Il campo dati email, keyPriv, keyPub deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

Questa pagina non prevede interazione con l'utente.

PiiListVM (implementa INotifyPropertyChanged)

È il controllore della pagina che mostra la lista dei servizi disponibili.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

piiList: è una lista di oggetti piiModel in databing con la vista.

Il campo dati email deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnClickPII(piiID:string)

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in PIIPage.
Parametri	• piiID: è una stringa che rappresenta la chiave della PII che ha generato l'evento.
Pseudo codice	`Navigation.PushAsync (new PIIPage(piiID));`
Note	

ServiceListVM (implementa INotifyPropertyChanged)

È il controllore della pagina che mostra la lista dei servizi disponibili.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

serviceList: è una lista di oggetti ServiceModel in databing con la vista.

Il campo dati email deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnClickService(piiID:string)

Descrizione	Il metodo ha il compito di eseguire le operazioni che portano al cambio della pagina in ServicePage.
Parametri	• serviceID: è una stringa che rappresenta la chiave del servizio che ha generato l'evento.
Pseudo codice	`Navigation.PushAsync (new ServicePage(serviceID));`
Note	

PIIPageVM (implementa INotifyPropertyChanged)

È il controllore della pagina che mostra la lista dei servizi disponibili.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

piiID: è una stringa che rappresenta la chiave della pii da visualizzare, deve essere passata tramite il costruttore dell'oggetto e messa in databing con la vista.

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

Pii: è un riferimento ad un oggetto PIIModel che contiene I dati relative alla PII identificata dal piiID fornito nel costruttore. Le informazioni sono visualizzate nella vista.

Il campo dati email deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnRemovePII(piiID:string)

Descrizione	Il metodo ha il compito rimuove la pii a cui fa riferimento la pagina.
Parametri	• piiID: è una stringa che rappresenta la chiave della PII che si vuole eliminare.
Pseudo codice	App.IWApplication.removePII(piiID); `Navigation.PushAsync (new PIIListPage(piiID));`
Note	

ServicePage (implementa INotifyPropertyChanged)

È il controllore della pagina che mostra la lista dei servizi disponibili.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

serviceID: è una stringa che rappresenta la chiave del Servizio da visualizzare, deve essere passata tramite il costruttore dell'oggetto e messa in databing con la vista.

email: è una stringa in databing, da visualizzare per mostrare l'active user attuale.

serviceModel: è un riferimento ad un oggetto ServiceModel che contiene I dati relative alla PII identificata dal serviceID fornito nel costruttore. Le informazioni sono visualizzate nella vista.

Il campo dati email deve avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

private void showQR(serviceID:string)

Descrizione	Il metodo ha il compito mostrare nell'applicazione il QR con le informazioni necessarie per effettuare il login al servizio.
Parametri	• serviceID: è una stringa che rappresenta la chiave della PII che si vuole eliminare.

Pseudo codice	Var services = App.currentSession.activeUser.getServiceList;
	Var ser = Services[serviceID];
	If ser != null {
	  s = serviceName.toString() + ser.requiredPII;
	}
	var qrwr = DependencyService.Get<Iqr>();
	`s = qrwr.GenQR(stringaInfo);`
	`imaInVista.Source = ImageSource.FromStream(()=>new MemoryStream(s));`

Note	

AddPIIPageVM (implementa INotifyPropertyChanged)

È il controllore della pagina di login.

Campi dati:

PropertyChanged: è un riferimento ad un oggetto Event (Xamarin).

piiName: è una stringa in databing con la form presente nella vista.

piiDesc: è una stringa in databing con la form presente nella vista.

SubmitCommand: e un riferimento ad un oggeto ICommand(Xamarin). Deve avere setter e getter pubblici.

I campi dati email, password devono avere i getter e setter tipici di C#. Il setter deve chiamare l'evento PropertyChanged.

Metodi:

public void OnSubmit()

Descrizione	Il metodo ha il compito definire le procedure da intraprendere per inserire la PII sia nella base di dati, sia nell'ITF
Parametri	

Pseudo codice	Var pii = new AbsPII(piiName, piiDesc);
	Var result = App.IWApplication.insertPII(pii);
	If result {
	  `Navigation.PushAsync (new PIIListPage());`

	} Else displayAllertError();
Note	