

LINEE GUIDA DI PROGRAMMAZIONE
SVILUPPO APPLICAZIONI “.NET”

TABELLA DELLE VERSIONI

Data	Versione	Descrizione	Cap. /Sez. modificati
Febbraio 2004	1.0	Nascita del documento	tutti

INDICE

1	Introduzione	4
1.1	Riferimenti	4
1.2	Definizioni ed Acronimi	4
2	Schemi Architetture di riferimento	5
3	Configurazione delle applicazioni	7
4	Gestione degli errori.....	9
4.1	Uso corretto di Try/Catch	9
5	Utilizzo di strong name	11
6	Autenticazione alle applicazioni ASP.NET	13

1 Introduzione

Obiettivo del presente documento di linee guida è quello di definire alcune regole di programmazione da rispettare durante la realizzazione di applicazioni WEB- based “.NET”.

1.1 Riferimenti

Codice	Titolo
N/A	Standard di programmazione: Integrazione con Login Server - Parte 2: applicazioni ASP.NET

1.2 Definizioni ed Acronimi

Definizione	Descrizione
strong name	"nome sicuro" nella traduzione italiana di Visual Studio
XSD	Acronimo usato per identificare una definizione XML Schema
XML	eXtensible Markup Language
http	HyperText Transfer Protocol
TCP/IP	Trasmission Control Protocol/Internet Protocol
COM	Component Object Model
CLSID	Class ID: un identificatore universale univoco (UUID) che identifica un componente COM. Ogni componente ha il suo CLSID nel Registry di Windows in modo da essere caricato da alter applicazioni.
GUID	Globally Inique IDentifier
C#	(C Sharp) Un nuovo linguaggio di sviluppo tra il C e Java
DB	Data Base
SOAP	Simple Object Access Protocol
GAC	Global Assembly Cache

2 Schemi Architetture di riferimento

In questo capitolo vengono descritti i principali schemi architetture a cui le applicazioni “.NET” possono far riferimento; si tratta comunque sempre di schemi architetture WEB-based.

Le possibili componenti architetture, necessarie all'esercizio dell'applicazione, devono poter essere installate su macchine differenti, al fine di garantire il requisito di scalabilità e il deployment sull'infrastruttura di rete del Ministero dell'Economia e delle Finanze.

Le componenti strutturali delle applicazioni devono rispettare la possibilità di essere suddivise su almeno tre layer differenti che comunicano attraverso il protocollo TCP/IP.

I tre layer dovranno rispettare la suddivisione funzionale nelle seguenti componenti:

- **Web Server:** dedicato alla gestione della interfaccia utente;
- **Application Server:** dedicato all'esecuzione delle transazioni applicative richieste dallo strato di interfaccia utente;
- **DB Server:** dedicato alla esecuzione delle transazioni dati richieste dallo strato applicativo.

La comunicazione tra i diversi strati deve avvenire utilizzando il protocollo TCP/IP, ed in particolare:

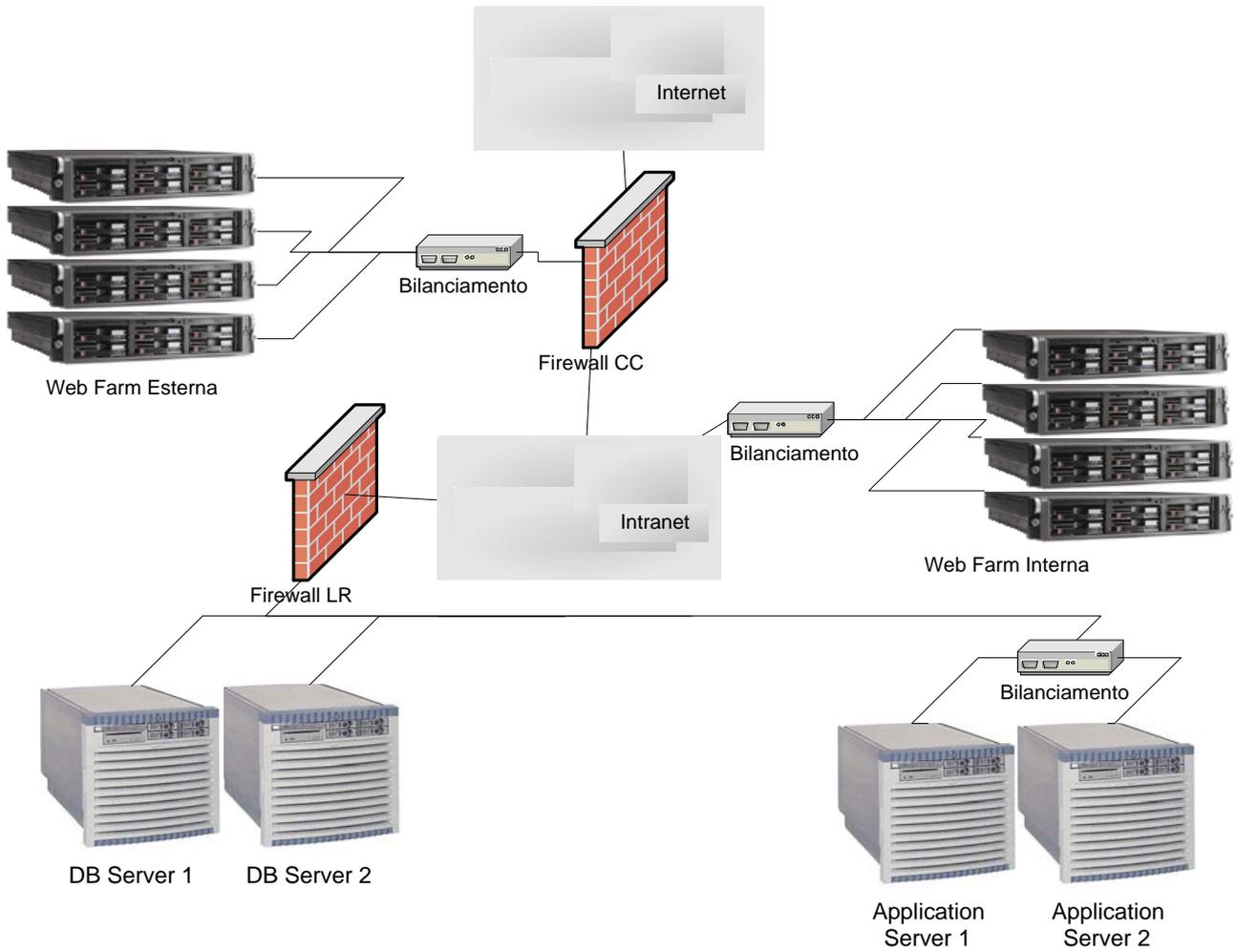
- Tra Web ed Application Server: possibilmente utilizzando il protocollo SOAP o le tecniche di Remoting proprietarie di .NET su protocollo HTTP;
- Tra Application e DB Server: su porte TCP/IP proprietarie.

Per applicazioni standard che coinvolgono solamente una base dati e fanno uso di un Application Server dedicato, i componenti applicativi dovranno essere sviluppati affinché non sia necessario l'uso della componente server MS Distributed Transaction Coordinator. Questa componente potrà essere utilizzata da applicazioni che richiedono invece il coinvolgimento di più Application Server o più DB Server all'interno della stessa transazione.

Le applicazioni, in base ai requisiti indicati sopra, dovranno essere calate sull'infrastruttura del Ministero dell'Economia e delle Finanze che sarà composta dalle seguenti componenti logiche:

- Web Farm in DMZ al CC (se aperta ad utenti internet/interdominio)
- Web Farm in DMZ a LR
- Application Server a LR

Le componenti sopraelencate, saranno dislocate nell'infrastruttura di rete come riportato sotto:



3 Configurazione delle applicazioni

Con il Framework .NET le configurazioni delle applicazioni sono basate su dei file *.config* che non sono altro che dei documenti XML a schema libero.

Per applicazioni che fanno uso di basi dati, servizi web, caselle di posta, ecc. , è necessario definire i parametri di configurazione dei servizi richiesti, affinché questi siano facilmente modificabili.

Il file di configurazione deve essere unico per tutti i servizi svolti dall'applicazione, in modo di semplificare le modifiche di configurazione dell'applicazione.

Solitamente questo tipo di configurazioni vengono inserite nel file *.config* nella sezione *appSettings* con delle coppie chiave/valore che rappresentano il nome e il valore dei nostri parametri di configurazione, ad esempio:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>
    <add key="sqlConnectionString"
      value="server=dbsrv;integrated security=SSPI;database=interno;" />
    <add key="security" value="Windows" />
    <add key="level" value="100" />
  </appSettings>

</configuration>
```

E' possibile utilizzare dei file di configurazione personalizzati, per risolvere le configurazioni più complesse di alcune applicazioni, basandosi sulla creazione di uno schema XSD che indica come deve essere costruito il file di configurazione. In questo modo otteniamo:

- o una configurazione più leggibile
- o possibilità di includere più configurazioni di diverse librerie nello stesso file
- o possibilità di validare la configurazione creata nel file *.config* all'interno di Visual Studio .NET grazie allo schema XSD
- o possibilità di creazione di classi dai valori serializzati nel file di configurazione

Ad esempio la configurazione descritta sopra può essere generata dal file XSD specificato sotto:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
  id="customConfiguration"
  targetNamespace="http://schemi.esempio.com/CustomConfiguration/"
  elementFormDefault="qualified"
  xmlns="http://schemi.esempio.com/CustomConfiguration/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="customConfiguration">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="sqlConnectionString" type="xsd:string" />
        <xsd:element name="security">
```

```
        <xsd:complexType>
            <xsd:attribute name="mode" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Windows" />
                        <xsd:enumeration value="Passport" />
                        <xsd:enumeration value="Custom" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
<xsd:attribute name="level" type="xsd:int" use="required" />
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Il file *.config* specificato sopra, quindi, potrà essere arricchito delle seguenti informazioni:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

    <appSettings>
        <add key="sqlConnectionString"
            value="server=dbsrv;integrated security=SSPI;database=interno;" />
        <add key="security" value="Windows" />
        <add key="level" value="100" />
    </appSettings>

    <customConfiguration
        xmlns="http://schemi.esempio.com/CustomConfiguration/"
        level="100">
        <sqlConnectionString>server=dbsrv;integrated
            security=SSPI;database=interno; </sqlConnectionString>
        <security mode="Windows" />
    </customConfiguration>

</configuration>
```

I valori così inseriti potranno essere letti dall'applicazione da una apposita classe, che potrà essere creata a partire dal file XSD, utilizzando il tool a riga di comando XSD.EXE con il parametro /c per generare la classe ed eventualmente /n (per creare un namespace dedicato).

Tutti i file di configurazione delle applicazioni dovranno sempre contenere i seguenti parametri:

- Configurazione della connessione al DB (IP, UserID, Password crittografata, Nome DB, porta)
- Configurazione di eventuali caselle di posta usate dall'applicazione (Nome del server, Nome della casella)
- Configurazione del server di Single Sign On
- Configurazione delle impostazioni di Logging degli Errori
- Configurazione delle impostazioni di Logging delle operazioni effettuate
- Configurazione di tutti gli eventuali parametri necessari.

4 Gestione degli errori

In questo capitolo sono indicate le linee guida da seguire per usare le eccezioni in un programma .NET.

L'uso della **vecchia sintassi "ON ERROR GOTO, ON ERROR RESUME"** di VB6, ancora supportata da VB.NET, **non deve essere più usata**, in quanto non consente prestazioni elevate e non permette una corretta gestione delle eccezioni.

In caso di applicazioni migrate da VB6 a VB.NET, sarà necessaria la riscrittura delle funzioni che usano la vecchia logica di gestione degli errori.

La gestione degli errori, fatta con On Error Goto e/o con Try/Catch, deve essere pensata solo per gestire le eccezioni, e non per sostituirsi a delle If che verificano le condizioni abituali.

Anche l'uso della nuova sintassi per la gestione degli errori, deve essere scritta in modo appropriato, ad esempio l'uso di un Try/Catch chiamato dentro un loop non è un'idea saggia, spesso il Try/Catch al di fuori del loop è più corretto anche dal punto di vista della logica applicativa.

Tutti gli errori devono essere tracciati in un apposito file di Log configurato nelle impostazioni di configurazione dell'applicazione. Il Log deve contenere tutte le informazioni di dettaglio dell'errore.

4.1 Uso corretto di Try/Catch

La gestione degli errori di VB6 è stata completamente rivista, in quanto il concetto di eccezione è stato inserito in tutti i livelli delle librerie .NET. E' buona regola intervenire per intercettare un'eccezione solo quando si ha la possibilità di fare qualcosa altrimenti, se lo scopo è solo quello di trasformare l'eccezione in un codice di errore, è inutile inserire l'intercettazione.

Ciò che è certo è che per intraprendere un'operazione e "riparare il danno" oppure per modificare il contenuto dell'eccezione, è necessario intervenire su funzioni di alto livello. In tutti gli altri casi, dove non sia possibile gestire realmente l'errore, il blocco Try/Catch non va inserito, poiché non farebbe altro che danneggiare le prestazioni. Si riporta, nel codice che segue, un esempio di un'eccezione che viene intercettata per annullare la transazione in corso:

```
Public Sub SaveData()  
    Dim tx As OleDbTransaction = connection.BeginTransaction()  
    Try  
        ' Codice che scrive su un database...  
        tx.Commit()  
    Catch e As Exception  
        tx.Rollback()  
    End Try  
End Sub
```

In questo caso si ha un esempio di "gestione" reale dell'errore: la transazione non è lasciata in uno stato "incompleto", viene invece completamente annullata e l'elaborazione può quindi proseguire. In tale esempio, però, viene intercettato qualunque tipo di eccezione diversamente da quanto avviene nella maggioranza dei casi, nei quali l'intento è quello di intraprendere un'azione diversa in funzione del tipo di errore registrato. La cosa migliore sarebbe quella di intercettare solo il tipo di eccezione che interessa.

Qualora siano presenti più blocchi Catch, la classe di eccezione più generica va specificata sempre dopo la classe più specifica.

In VB.NET occorre fare attenzione perché, al contrario di C#, il compilatore non segnala errori se non si rispetta questo vincolo:

```
Private Sub F1()  
    Try  
        F2(42, "test")  
    Catch e As ArgumentException  
        Console.WriteLine("Catch ArgumentException")  
        Console.WriteLine(e.Message)  
    Catch e As Exception  
        Console.WriteLine("Catch Exception")  
        Console.WriteLine(e.Message)  
    End Try  
End Sub
```

Intercettare la classe Exception, che è l'eccezione più generica, significa intercettare praticamente qualsiasi eccezione.

E' necessario che ogni thread di un programma abbia una propria gestione degli errori, in quanto la gestione delle eccezioni rimane all'interno del thread e non viene propagata dal chiamante.

L'unico modo per intercettare le eccezioni di thread esterni, è l'uso di un apposito evento (*AppDomain.UnhandledException*), nel quale convergono tutte le eccezioni non gestite da parte di qualsiasi thread di uno specifico application domain.

Oltre a risolvere il problema dei thread secondari, questo meccanismo consente di definire il metodo da agganciare ad un evento in un punto qualsiasi del programma. Lo scopo è quello di gestire gli errori al fine di produrne dei log.

Tale gestione, è obbligatoria in ogni applicazione e deve essere configurabile dal file di configurazione applicativo, attraverso uno o più flag. Ad esempio:

```
Shared Sub LogException(ByVal sender As Object, _  
                        ByVal e As UnhandledExceptionEventArgs)  
    If TypeOf e.ExceptionObject Is Exception Then  
        Dim ex As Exception  
        ex = CType(e.ExceptionObject, Exception)  
        Console.WriteLine("Unhandled exception {0} : {1}", _  
                          ex.GetType().Name, ex.Message)  
    Else  
        Console.WriteLine("Unhandled exception {0}", _  
                          e.ExceptionObject.GetType().Name)  
    End If  
End Sub  
  
Shared Sub Main()  
    Dim d As New DemoExceptions()  
    If ConfigurationSetting.AppSettings["Debug"] Then  
        AddHandler AppDomain.CurrentDomain.UnhandledException, _  
                  New UnhandledExceptionHandler(AddressOf LogException)  
    End If  
    d.ThreadDemo()  
End Sub
```

In questo caso, se il flag "Debug" è impostato, l'applicazione aggiunge un metodo per la gestione degli errori che ha la finalità, nel caso descritto, di scrivere messaggi di errore sulla Console.

5 Utilizzo di strong name

Uno strong name ("nome sicuro" nella traduzione italiana di Visual Studio) è un meccanismo di individuazione univoca degli assembly.

Lo strong name è composto da quattro parti:

- Nome
- Versione
- Cultura
- Public Key Token (hash della chiave pubblica, con cui si verifica la firma digitale)

Lo strong name assolve due compiti:

1. Garantire che il riferimento ad un assembly condiviso sia certo.
2. Garantire che l'assembly non sia stato modificato dopo il suo rilascio.

Le differenze rispetto a COM, dove l'univocità di un componente veniva definita da un GUID che identificava univocamente una classe, ha evidenziato i seguenti problemi:

- Versioni diverse dello stesso componente COM devono mantenere la compatibilità binaria con le versioni precedenti, perché era impossibile (almeno fino all'introduzione dei componenti side-by-side) installare diverse versioni dello stesso componente sulla stessa macchina.
- Le applicazioni che usano componenti COM non manifestano automaticamente una buona diagnostica degli errori quando il componente richiesto non è installato o è presente una versione incompatibile. In altre parole, il CLSID di un componente non fornisce molte informazioni sul componente stesso (come ad esempio il nome del componente).
- La gestione di versioni localizzate di un componente era demandata al produttore del componente; il sistema operativo non offriva un meccanismo predefinito per far convivere sulla stessa macchina diverse versioni localizzate dello stesso componente.

Quindi lo strong name definisce prima di tutto un riferimento univoco ad un assembly, come faceva il vecchio CLSID. Questo significa che gli utilizzatori di un assembly con strong name dovranno specificare 4 coordinate (nome, versione, cultura e public key token) per richiamare l'assembly desiderato.

Il secondo effetto che si ottiene fornendo uno strong name ad un assembly è di effettuare una verifica della firma digitale dell'assembly: qualsiasi modifica dei moduli che compongono l'assembly farà fallire tale controllo. Tale controllo (validazione dello strong name) avviene quando l'assembly viene installato nella GAC (Global Assembly Cache), oppure ogni volta che l'assembly viene caricato in memoria se è privato.

Un'altra conseguenza dello strong name è che un assembly con strong name può referenziare solo assembly dotati a loro volta di strong name, senza poter più referenziare assembly che ne sono privi. Questa limitazione assicura che, utilizzando un assembly dotato di strong name, non siano richiamati assembly privi di firma digitale. Un assembly con strong name deve essere validato, quindi anche quelli che referenzia devono ottemperare agli stessi requisiti.

I casi in cui devono essere usati gli strong name agli assembly sono i seguenti:

- **Libreria usata da molte applicazioni** – sia che si producano librerie da fornire a terze parti, sia che si producano per uso interno, il solo meccanismo di controllo delle versioni degli assembly è un motivo che giustifica lo strong name
- **Codice distribuito via internet/intranet** – un assembly può essere scaricato da remoto solo se è dotato di strong name; questo non garantisce l'identità dell'autore, ma semplifica il riconoscimento dell'autenticità dell'assembly richiesto (chi chiede l'assembly lo fa tramite lo strong name, e ottiene esattamente ciò che ha chiesto)
- **Installazione side-by-side** – se è necessario installare contemporaneamente più versioni dello stesso componente, è indispensabile installarle nella GAC, e per fare questo è necessario uno strong name
- **Ridirezione delle versioni** – per controllare la versione dell'assembly utilizzato da un'applicazione in maniera amministrativa, senza intervenire nel codice dell'applicazione stessa, lo strong name è indispensabile.
- **Garanzia da tampering (manomissioni)** – se un assembly ha uno strong name, la validazione dell'assembly fallisce se è stato modificato in seguito alla compilazione. Questa caratteristica è indispensabile quando si scarica un assembly da Internet.

Non è invece necessario usare uno strong name nei seguenti casi:

- **L'assembly è privato** – se l'assembly è usato solo come assembly privato da un'applicazione installata fisicamente sul disco locale di una macchina, il fatto di avere uno strong name implica una validazione dell'assembly ad ogni caricamento. Questo implica un certo overhead per l'operazione di lettura dell'intero file e di calcolo necessario alla verifica della firma digitale. Tutto ciò non significa che sia meglio non usare lo strong name sugli assembly privati, ma solo che non è strettamente necessario farlo.
- **È richiesta l'autenticazione dell'autore** – la firma digitale di uno strong name non garantisce l'identità dell'autore, quindi lo strong name non va considerato come mezzo di autenticazione. Questo non significa che lo strong name non debba essere usato in un caso simile, ma andrebbe affiancato all'uso di Authenticode. In un contesto simile lo strong name resta indispensabile per supportare i meccanismi di gestione delle versioni che abbiamo esaminato.

6 Autenticazione alle applicazioni ASP.NET

ASP.NET mette a disposizione tre meccanismi di autenticazione:

1. **Windows Authentication** in cui verranno sfruttati utenti e/o gruppi Windows
2. **Forms Authentication** in cui gli utenti verranno autenticati tramite cookie (persistente o meno) dopo aver fatto login tramite un form HTML completamente personalizzabile.
3. **Passport Authentication** che si appoggia al servizio centralizzato proposto da Microsoft per il single sign-on su internet.

Il Common Language Runtime (CLR) .NET fornisce un modello di sicurezza indipendente dal sistema operativo sottostante. Questo significa che un'applicazione .NET può utilizzare utenti e ruoli applicativi indipendenti dagli utenti e gruppi definiti nel sistema operativo.

Il meccanismo di autenticazione delle applicazioni ASP.NET deve essere integrato con lo strumento di Single Sign On del Ministero dell'Economia e delle Finanze specificato nel documento "*Standard di programmazione: Integrazione con Login Server - Parte 2: applicazioni ASP.NET*".

L'autenticazione delle applicazioni sarà basata quindi sul metodo di autenticazione di tipo "Forms". Questo tipo di autenticazione è quella che presenta le maggiori scoperture di sicurezza: la userid e la password viaggiano in chiaro sulla rete, quindi in molti casi è opportuno utilizzare il protocollo SSL per le operazioni di login.

Essendo l'autenticazione delegata ad un altro sistema (Oracle Login Server) è necessario che le nuove applicazioni definiscano la politica di autorizzazioni agli utenti. Esistono due strategie possibili di autorizzazione:

- o **Role based:** L'accesso alle funzionalità viene definito in base all'appartenenza dell'utente ad uno specifico ruolo. I ruoli vengono utilizzati per raggruppare una serie di utenti che condividono gli stessi privilegi all'interno dell'applicazione. L'applicazione non gestisce più direttamente gli utenti, ma l'associazione ruolo-funzione; ogni utente appartiene ad uno o più gruppi (ruoli) ereditandone i privilegi.
- o **Resource based:** Ogni specifica risorsa deve aver definita una propria ACL (Access Control List). L'accesso a tutte le risorse viene effettuato utilizzando i controlli di sicurezza propri di ciascun sottosistema (SO, RDBMS, etc...). Ovviamente questo approccio comporta grossi oneri gestionali.

Le nuove applicazioni devono adottare un approccio di tipo "Role based", solo tale approccio rappresenta la scelta più conveniente in contesti nei quali la scalabilità rappresenta un fattore chiave.

In conclusione, il modello necessario per lo sviluppo di nuove applicazioni per il MEF è il seguente:

- o L'autenticazione degli utenti è delegata all'Access Manager esterno (Oracle Login Server)
- o Gli utenti sono suddivisi in gruppi corrispondenti ai ruoli applicativi.
- o I gruppi fanno parte delle credenziali utente e vengono comunicati dall'access manager alle applicazioni assieme al nome e cognome
- o Le applicazioni autorizzano gli utenti all'utilizzo delle funzioni, in base ai ruoli di accesso
- o L'accesso alle risorse di back-end viene effettuato dalle applicazioni, utilizzando utenze specifiche.