SoluzionaApp

- Blazor WASM/PWA - Configurazione AppSettings per più clienti
- ☁ Gestione offline — Dati locali, coda e sincronizzazione
- Certificati HTTPS self-signed per IIS compatibili con Android e dispositivi Apple

Blazor WASM/PWA - Configurazione AppSettings per più clienti

Obiettivo

Gestire pubblicazioni diverse della stessa app Blazor WebAssembly/PWA per più clienti, evitando di modificare manualmente `appsettings.json` dopo la pubblicazione.

Nelle app Blazor WASM/PWA i file pubblicati vengono gestiti anche dal service worker e dal manifest degli asset:

```
service-worker-assets.js
```

Se si modifica manualmente un file statico dopo la pubblicazione, ad esempio `appsettings.json`, si possono creare problemi di cache, integrità o caricamento della PWA.

La soluzione corretta è:

```
1. mantenere un appsettings.json comune
2. creare un file appsettings per ogni cliente
3. tenere i file cliente fuori da wwwroot
4. usare un profilo di pubblicazione per cliente
5. fare copiare al .csproj solo il file del cliente pubblicato
```

---

1. Struttura cartelle consigliata

Nel progetto Blazor WASM, ad esempio `Soluziona.Apps.Magazzino`, usare questa struttura:

```
Soluziona.Apps.Magazzino
├── SettingClienti
│   ├── Comside
│   │   └── appsettings.Comside.json
│   └── Demo
│       └── appsettings.Demo.json
│
├── Properties
│   └── PublishProfiles
│       ├── Comside.pubxml
│       └── Demo.pubxml
│
└── wwwroot
    ├── appsettings.json
    ├── appsettings.Development.json
    ├── index.html
    ├── manifest.webmanifest
    ├── service-worker.js
    └── ...
```

Regola

Dentro `wwwroot` devono rimanere solo:

```
appsettings.json
appsettings.Development.json
```

I file dei clienti devono stare fuori da `wwwroot`, dentro:

```
SettingClienti\<NomeCliente>\appsettings.<NomeCliente>.json
```

Esempio:

```
SettingClienti\Comside\appsettings.Comside.json
SettingClienti\Demo\appsettings.Demo.json
```

Non mettere dentro `wwwroot`:

```
appsettings.Comside.json
appsettings.Demo.json
```

altrimenti rischiano di essere pubblicati tutti insieme.

---

2. `appsettings.json` comune

Il file:

```
wwwroot\appsettings.json
```

deve contenere solo i valori comuni a tutti i clienti.

Esempio:

```
{
  "App": {
    "NomeApp": "Liste di Carico",
    "SottoTitoloApp": "Gestione preparazione ordini",
    "Versione": "1.0.6",
    "Build": "2026.09.09.006",
```

```
    "Ambiente": "Production",
    "PaginaIniziale": "/calendario"
  },
  "Api": {
    "TimeoutSecondi": 30
  },
  "Offline": {},
  "SyncListeDiCarico": {
    "IntervalloSecondi": 30,
    "GiorniIndietro": 15,
    "GiorniAvanti": 15,
    "EseguiAllAvvio": true
  }
}
```

Nota

In questo file può mancare:

```
"BaseUrl"
```

perché viene valorizzato dal file cliente, ad esempio:

```
appsettings.Comside.json
```

A runtime Blazor caricherà:

```
appsettings.json
+
appsettings.Comside.json
```

3. `appsettings.Development.json`

Il file:

```
wwwroot\appsettings.Development.json
```

viene usato per lo sviluppo locale.

Esempio:

```
{
  "App": {
    "IdApplicazione": "17630f17-7626-416f-961b-486fbd64fd4f",
    "CodiceApplicazione": "APPLISTEDICARICO",
    "CodiceAzienda": "3",
    "Ambiente": "Development"
  },
  "Api": {
    "BaseUrl": "https://localhost:7021/",
    "TimeoutSecondi": 30
  }
}
```

Questo file non deve essere pubblicato in produzione.

4. AppSettings per cliente

Demo

Percorso:

```
SettingClienti\Demo\appsettings.Demo.json
```

Contenuto:

```
{
  "App": {
    "IdApplicazione": "17630f17-7626-416f-961b-486fbd64fd4f",
    "CodiceApplicazione": "APPLISTEDICARICO",
    "CodiceAzienda": "3",
    "Ambiente": "Production"
  },
  "Api": {
    "BaseUrl": "https://soluzionacoreapi.soluziona.cloud:443/",
    "TimeoutSecondi": 30
  }
}
```

Comside

Percorso:

```
SettingClienti\Comside\appsettings.Comside.json
```

Contenuto:

```
{
  "App": {
    "IdApplicazione": "90208117-20b4-446c-a771-42bce3fb63c5",
    "CodiceApplicazione": "ListeDiCarico",
    "CodiceAzienda": "59",
    "Ambiente": "Production"
  },
  "Api": {
    "BaseUrl": "https://192.168.232.122:4433/",
    "TimeoutSecondi": 30
  }
}
```

---

5. Modifica del `.csproj`

Aprire il file `.csproj` del progetto Blazor WASM, ad esempio:

```
Soluziona.Apps.Magazzino.csproj
```

La configurazione deve:

```
1. escludere SettingClienti dagli item automatici del SDK
2. escludere eventuali appsettings ambiente da wwwroot
3. copiare il file cliente corretto dentro wwwroot prima della build/publish
4. includere esplicitamente il file copiato come Content
```

5.1 Esclusione file cliente e appsettings ambiente

Nel primo `PropertyGroup` aggiungere:

```
<DefaultItemExcludes>$(DefaultItemExcludes);SettingClienti\**\*;wwwroot\appsettings.*.json</De
faultItemExcludes>
```

Esempio:

```
<PropertyGroup>
  <TargetFramework>net10.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
  <OverrideHtmlAssetPlaceholders>true</OverrideHtmlAssetPlaceholders>
  <ServiceWorkerAssetsManifest>service-worker-assets.js</ServiceWorkerAssetsManifest>

  <!-- Esclude i file cliente dagli item automatici del .NET SDK -->

<DefaultItemExcludes>$(DefaultItemExcludes);SettingClienti\**\*;wwwroot\appsettings.*.json</De
faultItemExcludes>
</PropertyGroup>
```

Questa riga esclude:

```
SettingClienti\...
wwwroot\appsettings.Demo.json
wwwroot\appsettings.Comside.json
wwwroot\appsettings.Development.json
```

Non esclude:

```
wwwroot\appsettings.json
```

perché il file comune deve sempre essere pubblicato.

5.2 Reincludere `appsettings.Development.json`

Poiché `wwwroot\appsettings.*.json` viene escluso dagli item automatici, `appsettings.Development.json` va reincluso esplicitamente, ma senza pubblicarlo.

```
<Content Include="wwwroot\appsettings.Development.json">
  <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  <CopyToPublishDirectory>Never</CopyToPublishDirectory>
</Content>
```

5.3 Target per preparare l’appsettings cliente

Aggiungere questo target prima di:

```
</Project>
```

```
<Target Name="PreparaAppSettingsCliente"
        BeforeTargets="AssignTargetPaths"
        Condition="'$(WasmApplicationEnvironmentName)' != ''
                and '$(WasmApplicationEnvironmentName)' != 'Development'
                and
Exists('$(MSBuildProjectDirectory)\SettingClienti\$(WasmApplicationEnvironmentName)\appsetting
s.$(WasmApplicationEnvironmentName).json')">

  <Message Importance="High"
           Text="Preparo appsettings cliente:
SettingClienti\$(WasmApplicationEnvironmentName)\appsettings.$(WasmApplicationEnvironmentName)
.json" />

  <!-- Elimina eventuali appsettings cliente generati da build precedenti -->
  <ItemGroup>
    <AppSettingsClienteInWwwRoot
Include="$(MSBuildProjectDirectory)\wwwroot\appsettings.*.json"

Exclude="$(MSBuildProjectDirectory)\wwwroot\appsettings.Development.json" />
  </ItemGroup>

  <Delete Files="@(AppSettingsClienteInWwwRoot)" />

  <!-- Copia il file cliente dentro wwwroot prima della build/publish -->
  <Copy
SourceFiles="$(MSBuildProjectDirectory)\SettingClienti\$(WasmApplicationEnvironmentName)\appse
ttings.$(WasmApplicationEnvironmentName).json"

DestinationFiles="$(MSBuildProjectDirectory)\wwwroot\appsettings.$(WasmApplicationEnvironmentN
ame).json"
        SkipUnchangedFiles="false" />

  <!-- Include esplicitamente l'appsettings cliente generato come Content -->
  <ItemGroup>
    <Content Include="wwwroot\appsettings.$(WasmApplicationEnvironmentName).json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    </Content>
  </ItemGroup>
```

```
</Target>
```

---

6. `.csproj` completo finale

```
<Project Sdk="Microsoft.NET.Sdk.BlazorWebAssembly">

  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <OverrideHtmlAssetPlaceholders>true</OverrideHtmlAssetPlaceholders>
    <ServiceWorkerAssetsManifest>service-worker-assets.js</ServiceWorkerAssetsManifest>

    <!-- Esclude i file cliente dagli item automatici del .NET SDK -->

<DefaultItemExcludes>$(DefaultItemExcludes);SettingClienti\**\*;wwwroot\appsettings.*.json</De
faultItemExcludes>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="DevExpress.Blazor" Version="26.1.4" />
    <PackageReference Include="DevExpress.Blazor.Themes" Version="26.1.4" />
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly" Version="10.0.11"
/>
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.DevServer"
Version="10.0.11" PrivateAssets="all" />
    <PackageReference Include="Microsoft.Extensions.Http" Version="10.0.11" />
    <PackageReference Include="System.Security.Cryptography.Xml" Version="10.0.11" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Soluziona.Apps.UI\Soluziona.Apps.UI.csproj" />
    <ProjectReference Include="..\Soluziona.Apps.Utility\Soluziona.Apps.Core.csproj" />
```

```
    <ProjectReference Include="..\SoluzionaApps.ApiClient\Soluziona.Apps.ApiClient.csproj" />
  </ItemGroup>

  <ItemGroup>
    <ServiceWorker Include="wwwroot\service-worker.js" PublishedContent="wwwroot\service-
worker.published.js" />
  </ItemGroup>

  <ItemGroup>
    <Content Update="wwwroot\web.config">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>

    <!-- Development escluso dai Content automatici, quindi lo reincludiamo esplicitamente -->
    <Content Include="wwwroot\appsettings.Development.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
      <CopyToPublishDirectory>Never</CopyToPublishDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <Folder Include="Properties\PublishProfiles\" />
  </ItemGroup>

  <Target Name="PreparaAppSettingsCliente"
          BeforeTargets="AssignTargetPaths"
          Condition="'$(WasmApplicationEnvironmentName)' != ''
                  and '$(WasmApplicationEnvironmentName)' != 'Development'
                  and
Exists('$(MSBuildProjectDirectory)\SettingClienti\$(WasmApplicationEnvironmentName)\appsetting
s.$(WasmApplicationEnvironmentName).json')">

    <Message Importance="High"
             Text="Preparo appsettings cliente:
SettingClienti\$(WasmApplicationEnvironmentName)\appsettings.$(WasmApplicationEnvironmentName)
.json" />

    <!-- Elimina eventuali appsettings cliente generati da build precedenti -->
    <ItemGroup>
      <AppSettingsClienteInWwwRoot
```

```
Include="$(MSBuildProjectDirectory)\wwwroot\appsettings.*.json"

Exclude="$(MSBuildProjectDirectory)\wwwroot\appsettings.Development.json" />
    </ItemGroup>

    <Delete Files="@(AppSettingsClienteInWwwRoot)" />

    <!-- Copia il file cliente dentro wwwroot prima della build/publish -->
    <Copy
SourceFiles="$(MSBuildProjectDirectory)\SettingClienti\$(WasmApplicationEnvironmentName)\appse
ttings.$(WasmApplicationEnvironmentName).json"

DestinationFiles="$(MSBuildProjectDirectory)\wwwroot\appsettings.$(WasmApplicationEnvironmentN
ame).json"
          SkipUnchangedFiles="false" />

    <!-- Include esplicitamente l'appsettings cliente generato come Content -->
    <ItemGroup>
      <Content Include="wwwroot\appsettings.$(WasmApplicationEnvironmentName).json">
        <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
        <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
      </Content>
    </ItemGroup>

  </Target>

</Project>
```

Nota importante per Git

Il target copia automaticamente il file cliente dentro `wwwroot`.

Esempio:

```
SettingClienti\Comside\appsettings.Comside.json
→ wwwroot\appsettings.Comside.json
```

Questo file in `wwwroot` è generato automaticamente e non deve essere committato.

Aggiungere al `.gitignore` in fondo:

```
# AppSettings cliente generati automaticamente dal publish
**/wwwroot/appsettings.*.json

# Mantiene versionato appsettings.Development.json
!**/wwwroot/appsettings.Development.json
```

`wwwroot\appsettings.json` non viene escluso da questa regola, quindi rimane versionato normalmente.

7. Profili di pubblicazione

I profili vanno creati da Visual Studio con:

```
Tasto destro sul progetto
→ Publish
→ New profile
→ Folder
```

Dopo la creazione, modificare i file `.pubxml`.

Profilo Demo

Percorso:

```
Properties\PublishProfiles\Demo.pubxml
```

Contenuto:

```
<?xml version="1.0" encoding="utf-8"?>
<Project>
  <PropertyGroup>
    <DeleteExistingFiles>true</DeleteExistingFiles>
    <ExcludeApp_Data>false</ExcludeApp_Data>
    <LaunchSiteAfterPublish>true</LaunchSiteAfterPublish>

    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>

    <PublishProvider>FileSystem</PublishProvider>
    <PublishUrl>C:\Repos\Pubblicati\AppListeDiCarico\Demo</PublishUrl>
    <WebPublishMethod>FileSystem</WebPublishMethod>
    <_TargetId>Folder</_TargetId>

    <WasmApplicationEnvironmentName>Demo</WasmApplicationEnvironmentName>
  </PropertyGroup>
</Project>
```

Profilo Comside

Percorso:

```
Properties\PublishProfiles\Comside.pubxml
```

Contenuto:

```
<?xml version="1.0" encoding="utf-8"?>
<Project>
  <PropertyGroup>
    <DeleteExistingFiles>true</DeleteExistingFiles>
```

```
    <ExcludeApp_Data>false</ExcludeApp_Data>
    <LaunchSiteAfterPublish>true</LaunchSiteAfterPublish>

    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>

    <PublishProvider>FileSystem</PublishProvider>
    <PublishUrl>C:\Repos\Pubblicati\AppListeDiCarico\Comside</PublishUrl>
    <WebPublishMethod>FileSystem</WebPublishMethod>
    <_TargetId>Folder</_TargetId>

    <WasmApplicationEnvironmentName>Comside</WasmApplicationEnvironmentName>
  </PropertyGroup>
</Project>
```

---

8. Proprietà fondamentale: `WasmApplicationEnvironmentName`

La riga più importante del profilo è questa:

```
<WasmApplicationEnvironmentName>Comside</WasmApplicationEnvironmentName>
```

oppure:

```
<WasmApplicationEnvironmentName>Demo</WasmApplicationEnvironmentName>
```

Questa proprietà decide quale ambiente Blazor deve usare.

Esempio:

```
WasmApplicationEnvironmentName = Comside
→ carica appsettings.json
→ carica appsettings.Comside.json
```

```
WasmApplicationEnvironmentName = Demo
→ carica appsettings.json
→ carica appsettings.Demo.json
```

Il valore deve combaciare esattamente con:

```
nome cartella cliente
nome file appsettings
```

Esempio corretto:

```
Comside
→ SettingClienti\Comside\appsettings.Comside.json
```

Esempio errato:

```
COMSIDE
→ SettingClienti\Comside\appsettings.Comside.json
```

---

9. Nota su `App:Ambiente`

Dentro gli appsettings cliente si può lasciare:

```
"Ambiente": "Production"
```

Questo valore è una configurazione applicativa interna.

Non decide quale file `appsettings.*.json` viene caricato.

Il file caricato viene deciso da:

```
<WasmApplicationEnvironmentName>Comside</WasmApplicationEnvironmentName>
```

non da:

```
"Ambiente": "Production"
```

?? Gestione offline — Dati locali, coda e sincronizzazione

Questa pagina descrive come utilizzare `IndexedDbLocalDataStoreService`, `CodaOfflineService` e `OfflineCrudServiceBase<TEntita>` per gestire dati e operazioni offline nelle applicazioni Soluziona.

Il flusso prevede il salvataggio sul dispositivo e il successivo invio al gestionale attraverso una coda persistente.

> □□ **Salvataggio locale e salvataggio sul server sono distinti**
> Un'operazione accodata è disponibile sul dispositivo, ma deve ancora essere confermata dal server.

? Responsabilità dei servizi

Componente	Responsabilità
`IndexedDbLocalDataStoreService`	Salva, legge ed elimina dati nel database locale del browser.
`CodaOfflineService`	Memorizza le operazioni da inviare, il loro collegamento alle entità e gli errori di sincronizzazione.
`OfflineCrudServiceBase<TEntita>`	Fornisce lettura e salvataggio locali e gestisce i diversi casi di eliminazione offline.
`SyncService`	Esegue le chiamate HTTP presenti in coda.
`SyncMonitorService`	Richiede la sincronizzazione quando cambia la connessione o viene modificata la coda, se il dispositivo risulta online.
`ISyncSuccessHandler`	Aggiorna i dati locali dopo una risposta positiva del server, secondo le regole del dominio.

? Flusso generale

```
L’utente esegue un’operazione
            ↓
Il servizio applicativo salva l’entità in IndexedDB
            ↓
Accoda il comando da inviare al server
            ↓
La UI può mostrare il dato locale come “Da sincronizzare”
            ↓
SyncService invia il comando quando viene avviata la sincronizzazione
            ↓
Il server risponde positivamente
            ↓
L’handler aggiorna l’entità locale
            ↓
L’operazione viene marcata come sincronizzata e rimossa dalla coda
```

Questo modello può essere utilizzato anche quando il dispositivo è già online: il servizio applicativo salva e accoda, mentre il monitor può avviare subito l’invio.

> **i L’offline va implementato per ciascuna funzionalità**
> La presenza dei servizi non rende automaticamente offline tutte le chiamate API dell’app.

? IndexedDbLocalDataStoreService

È il servizio di accesso ai dati locali. Serializza gli oggetti in JSON e richiama le funzioni JavaScript di `soluzionaIndexedDb`.

Metodi disponibili

Metodo	Utilizzo
`SalvaAsync<T>(storeName, key, value)`	Inserisce o sostituisce il valore associato alla chiave.
`RicercaByKeyAsync<T>(storeName, key)`	Recupera un oggetto; restituisce il valore predefinito se assente.
`ElencoAsync<T>(storeName)`	Legge gli oggetti dello store logico.

Metodo	Utilizzo
`EliminaAsync(storeName, key)`	Elimina un singolo oggetto.
`PulisciStoreAsync(storeName)`	Elimina tutti gli oggetti dello store logico.

Store e chiavi

Nell’implementazione JavaScript di Timbratura:

Elemento	Valore
Database	`SoluzionaAppsDb`
Object store fisico	`local_data`
Identificativo del record	`storeName::key`

Il parametro `storeName` rappresenta quindi un **raggruppamento logico** all’interno dello stesso object store.

Esempio di utilizzo

```
string store = TimbraturaLibrary.STORE_RICHIESTA;
string key = richiesta.Id.ToString();

// Salvataggio locale
await localDataStore.SalvaAsync(store, key, richiesta, cancellationToken);

// Lettura per chiave
CustomRichiesta? salvata = await localDataStore.RicercaByKeyAsync<CustomRichiesta>(store, key,
cancellationToken);

// Lettura elenco
List<CustomRichiesta> elenco = await localDataStore.ElencoAsync<CustomRichiesta>(store,
cancellationToken);
```

> □□ **Salvare in IndexedDB non accoda alcuna chiamata**
> L’invio al server deve essere predisposto esplicitamente tramite `CodaOfflineService`.

? CodaOfflineService

La coda viene memorizzata nello store logico **`offline_queue`**.

Ogni elemento descrive un’operazione da eseguire verso il server.

Informazioni principali

Campo	Significato
`Id`	Identificativo dell’elemento in coda.
`IdOperazioneClient`	Identificativo dell’operazione, utile per riconoscere i tentativi ripetuti.
`TipoOperazione`	Identifica la funzione applicativa e gli handler da eseguire.
`AzioneCoda`	Classifica l’azione: creazione, modifica, eliminazione o personalizzata.
`Endpoint`	Percorso API da chiamare.
`MetodoHttp`	Metodo HTTP da utilizzare.
`PayloadJson`	Corpo della richiesta serializzato.
`StoreNameEntitaLocale`	Store dell’entità collegata.
`KeyEntitaLocale`	Chiave dell’entità collegata.
`NumeroTentativi`	Numero di tentativi falliti registrati.
`UltimoErrore`	Dettaglio dell’ultimo errore.
`IsSincronizzato`	Indica se l’operazione è stata completata.

Accodare un’operazione

Estratto del flusso utilizzato per le richieste:

```
await CodaOfflineService.AccodaAsync(
    tipoOperazione: TimbraturaOperazioniOffline.RichiestaCreate,
    endpoint: "api/Richiesta/Nuova",
    payload: request,
    storeNameEntitaLocale: TimbraturaLibrary.STORE_RICHIESTA,
    keyEntitaLocale: richiestaLocale.Id.ToString(),
    azioneCoda: AzioniCodaOffline.Create,
    cancellationToken: cancellationToken);
```

Il collegamento tramite **store e chiave** permette di recuperare tutte le operazioni relative alla stessa entità.

> ⚠ **Azione logica e metodo HTTP sono diversi**
> `AccodaAsync` imposta attualmente `MetodoHttp = "POST"`. Passare `AzioniCodaOffline.Delete` classifica l'operazione come eliminazione, ma non la trasforma in una richiesta HTTP `DELETE`.

Metodi di gestione

Metodo	Utilizzo
`GetPendentiAsync()`	Restituisce le operazioni non sincronizzate, ordinate per creazione.
`GetPendentiPerEntitaAsync(...)`	Recupera le operazioni collegate a un'entità.
`EsisteCreatePendentePerEntitaAsync(...)`	Verifica la presenza di una creazione pendente.
`RimuoviPendentiPerEntitaAsync(...)`	Rimuove tutte o alcune azioni pendenti dell'entità.
`SetAsSincronizzatoAsync(...)`	Registra il completamento dell'operazione.
`SetAsFallitoAsync(...)`	Incrementa i tentativi e memorizza l'errore.
`PulisciSincronizzatiAsync()`	Elimina dalla coda le operazioni completate.

Le modifiche alla coda generano l'evento **`CodaModificata`**, utilizzabile dal monitor e dalla UI.

? OfflineCrudServiceBase

La classe base richiede un'entità che implementi **`IEntitaOffline`**.

```
public interface IEntitaOffline
{
    Guid Id { get; set; }
    Guid? IdServer { get; set; }
    EStatoSyncLocale StatoSyncLocale { get; set; }
    bool IsEliminataLocalmente { get; set; }
}
```

Proprietà	Significato
`Id`	Identificativo usato localmente.
`IdServer`	Identificativo del gestionale, se già disponibile.
`StatoSyncLocale`	Stato della sincronizzazione dell'entità.
`IsEliminataLocalmente`	Segnala che l'entità deve essere esclusa dalla visualizzazione.

Stati disponibili

Stato	Significato
`Sincronizzata`	Entità allineata al server.
`DaCreare`	Creazione ancora da sincronizzare.
`DaModificare`	Modifica ancora da sincronizzare.
`DaEliminare`	Eliminazione ancora da sincronizzare.
`ErroreSync`	Stato disponibile per rappresentare un errore di sincronizzazione.

> **i Gli stati non vengono tutti gestiti automaticamente**
> Il servizio concreto e gli handler devono mantenere aggiornato lo stato. Ad esempio, un errore della coda non imposta automaticamente `ErroreSync` sull'entità.

Cosa implementa la classe base

- `RicercaLocaleByIdAsync`: cerca l'entità nello store configurato.
- `SalvaLocaleAsync`: salva l'entità usando, per impostazione predefinita, `Id.ToString()` come chiave.
- `EliminaAsync`: applica le regole di eliminazione offline.

Il servizio derivato deve definire:

```
protected override string StoreName => TimbraturaLibrary.STORE_RICHIESTA;


protected override Task AccodaEliminazioneAsync( entita, cancellationToken = default)
{
    // Accodamento del comando specifico del dominio.
```

```
}
```

Creazione, modifica, lettura dal server e gestione dei conflitti restano responsabilità del servizio applicativo.

?? Come funziona l’eliminazione offline

Caso 1 — Entità non ancora sincronizzata

La classe base considera questo caso quando si verifica almeno una condizione:

- Esiste una `Create` pendente.
- Lo stato è `DaCreare`.
- `IdServer` è `null`.

L’eliminazione:

1. Rimuove tutte le operazioni pendenti collegate.
2. Elimina fisicamente l’entità locale.
3. Non accoda un comando di eliminazione.

```
Creazione locale → CREATE in coda → Eliminazione prima dell’invio
                                          ↓
                                Rimozione del dato e della coda
```

Caso 2 — Entità già presente sul server

L’eliminazione:

1. Rimuove le eventuali `Update` pendenti.
2. Imposta `StatoSyncLocale = DaEliminare`.
3. Imposta `IsEliminataLocalmente = true`.
4. Salva l’entità locale.
5. Chiama `AccodaEliminazioneAsync`.

Il servizio concreto prepara il comando per il gestionale. Dopo la conferma, l’handler può eliminare definitivamente il dato locale.

> **□□ La UI deve filtrare le entità eliminate localmente**
> La classe base imposta il flag, ma non modifica direttamente le liste visualizzate.

```
var visibili = elenco
    .Where(x => !x.IsEliminataLocalmente)
    .ToList();
```

? Invio e gestione degli esiti

`SyncService.SincronizzaAsync()`:

1. Verifica che non sia già in corso una sincronizzazione nella stessa istanza.
2. Controlla la connessione.
3. Recupera le operazioni pendenti.
4. Seleziona quelle con meno di **5 tentativi falliti**.
5. Le elabora in ordine di creazione.
6. Invia le richieste tramite il client autenticato **`SoluzionaApi`**.
7. Esegue gli handler compatibili quando la risposta HTTP è positiva.
8. Marca le operazioni riuscite e le rimuove dalla coda.

In caso di errore, l'operazione rimane pendente con `NumeroTentativi` e `UltimoErrore` aggiornati.

> ⚠ **Dopo 5 fallimenti l'operazione resta in coda, ma viene esclusa dai successivi invii**
> Occorre prevedere una gestione applicativa per diagnosi e ripristino. Il servizio non applica automaticamente un nuovo ciclo illimitato di tentativi.

L'evento **`SyncCompletata`** indica che il ciclo non ha prodotto errori tra gli elementi elaborati: non garantisce che la coda sia vuota, perché possono restare elementi esclusi per numero di tentativi.

? Handler dopo la sincronizzazione

Gli handler implementano **`ISyncSuccessHandler`** e vengono selezionati in base a `TipoOperazione`.

Sono il punto in cui gestire:

- L'identificativo restituito dal server.

- L’allineamento dei dati locali alla risposta.
- Il passaggio a `Sincronizzata`.
- La rimozione locale dopo un’eliminazione confermata.

Nel progetto Timbratura sono registrati:

```
services.AddScoped<ISyncSuccessHandler,RichiestaCreateSyncSuccessHandler>();
services.AddScoped<ISyncSuccessHandler,RichiestaDeleteSyncSuccessHandler>();
```

> ◫ **Registrare gli handler è parte dell’integrazione**
> Senza un handler compatibile, `SyncService` può comunque considerare riuscita la chiamata HTTP e rimuoverla dalla coda, lasciando il dato locale non allineato.

?? Come integrare una nuova entità

1. **Definisci il modello locale** implementando `IEntitaOffline`.
2. **Scegli uno store logico** e una chiave coerente.
3. **Crea il servizio applicativo** derivando da `OfflineCrudServiceBase<TEntita>`.
4. **Implementa la creazione locale** con ID locale, `IdServer = null` e stato `DaCreare`.
5. **Salva e accoda** il DTO destinato all’API, collegandolo a store e chiave.
6. **Implementa l’accodamento dell’eliminazione** per le entità già sincronizzate.
7. **Implementa e registra gli handler** di successo.
8. **Integra la UI** con dati locali, indicatori di sincronizzazione e filtro delle eliminazioni.
9. **Avvia il monitor** nel ciclo di vita dell’app e fermalo quando non serve più.

Per richiedere esplicitamente un ciclo di sincronizzazione:

```
var esiti = await syncService.SincronizzaAsync(cancellationToken);
```

`SyncMonitorService.Avvia()` sottoscrive gli eventi; non esegue da solo un invio iniziale e non introduce un timer periodico.

? Aspetti da gestire nell’applicazione

Aspetto	Indicazione
Idempotenza	Il server deve riconoscere `IdOperazioneClient` per evitare duplicazioni in caso di reinvio. La sola presenza dell’ID nel client non basta.

Aspetto	Indicazione
Atomicità	Salvataggio dell’entità e accodamento sono operazioni separate: gestisci il caso in cui riesca soltanto la prima.
Operazioni già in invio	Rimuovere un elemento dalla coda non annulla una richiesta HTTP già partita. Coordinare eliminazione e sincronizzazione richiede attenzione.
Aggiornamento delle liste	Unisci i dati server con quelli locali pendenti, evitando di perdere creazioni offline o ripristinare elementi marcati per eliminazione.
Cambio utente	La coda viene inviata con la sessione corrente: gestisci l’appartenenza dei dati a utente e azienda prima di consentire cambi di account.
Persistenza locale	IndexedDB conserva i dati nel browser; la cancellazione dei dati del sito può rimuovere anche le operazioni non inviate.
Disponibilità offline dell’app	IndexedDB conserva i dati. Il caricamento dell’app senza rete dipende anche dalla configurazione PWA e dal service worker.

> □□ **Messaggi corretti per l’utente**
> Dopo l’accodamento usa una conferma come **“Salvato sul dispositivo, in attesa di sincronizzazione”**. Riserva **“Sincronizzato”** alla conferma effettiva del server.

??? Guida per lo sviluppatore — Esempio generico

L’esempio seguente mostra come gestire offline una semplice **nota**, dalla creazione locale all’eliminazione.

I nomi degli endpoint e i DTO sono illustrativi: devono essere adattati alle API del progetto.

1. Definisci il modello locale

Il modello locale contiene i dati applicativi e le informazioni necessarie alla sincronizzazione.

```
public sealed class CustomNotaLocale : IEntitaOffline
{
    public Guid Id { get; set; }
    public Guid? IdServer { get; set; }

    public EStatoSyncLocale StatoSyncLocale { get; set; }
    public bool IsEliminataLocalmente { get; set; }

    public string Testo { get; set; } = string.Empty;
}
```

Definisci separatamente i DTO scambiati con il server:

```
public sealed class ReqCreaNota : ReqOperazioneOfflineBase
{
    public string Testo { get; set; } = string.Empty;
}

public sealed class ReqEliminaNota : ReqOperazioneOfflineBase
{
    public Guid IdNota { get; set; }
}

public sealed class ResNota
{
    public Guid Id { get; set; }
    public string Testo { get; set; } = string.Empty;
}
```

> **□□ Modello locale e DTO hanno scopi diversi**
> Il modello locale serve alla UI e alla gestione offline. Il DTO contiene i dati richiesti dall'API. Gli stati di sincronizzazione non devono necessariamente essere inviati al server.

2. Definisci store e tipi di operazione

Usa costanti condivise tra servizio e handler.

```
public static class {NomeApp}Library
{
    public const string Store = "note";

    public const string Create = "NOTA_CREATE";
    public const string Delete = "NOTA_DELETE";
}
```

`Store` identifica il raggruppamento dei dati locali. `Create` e `Delete` permettono al sincronizzatore di individuare gli handler appropriati.

3. Implementa il servizio applicativo

Il servizio eredita la gestione dell'eliminazione da `OfflineCrudServiceBase`.

```
public sealed class NotaService : OfflineCrudServiceBase<NotaLocale>
{
    protected override string StoreName => NotaOffline.Store;

    public NotaService(IIndexedDbLocalDataStoreService localDataStore, ICodaOfflineService
codaOfflineService) : base(localDataStore, codaOfflineService)
    {
    }

    public async Task<CustomNotaLocale> CreaAsync(string testo, CancellationToken
cancellationToken = default)
    {
        if (string.IsNullOrWhiteSpace(testo))
            throw new ArgumentException("Inserisci il testo della nota.");

        var nota = new CustomNotaLocale
        {
            Id = Guid.NewGuid(),
            IdServer = null,
            Testo = testo.Trim(),
            StatoSyncLocale = EStatoSyncLocale.DaCreare,
            IsEliminataLocalmente = false
        };

        var request = new ReqCreaNota
```

```
        {
            IdOperazioneClient = Guid.NewGuid(),
            Testo = nota.Testo
        };

        // 1. Rende il dato disponibile sul dispositivo.
        await SalvaLocaleAsync(nota, cancellationToken);

        // 2. Predispone l'invio al server.
        await CodaOfflineService.AccodaAsync(
            tipoOperazione: NotaOffline.Create,
            endpoint: "api/Note/Crea",
            payload: request,
            storeNameEntitaLocale: StoreName,
            keyEntitaLocale: GetKey(nota.Id),
            azioneCoda: AzioniCodaOffline.Create,
            cancellationToken: cancellationToken);

        return nota;
    }

    public async Task<List<CustomNotaLocale>> ElencoLocaleAsync(CancellationToken
cancellationToken = default)
    {
        var elenco = await LocalDataStore.ElencoAsync<NotaLocale>(StoreName,
cancellationToken);

        return elenco.Where(x => !x.IsEliminataLocalmente).ToList();
    }

    protected override async Task AccodaEliminazioneAsync(NotaLocale entita, CancellationToken
cancellationToken = default)
    {
        if (!entita.IdServer.HasValue || entita.IdServer.Value == Guid.Empty)
        {
            throw new InvalidOperationException( "Identificativo server della nota non
disponibile.");
        }

        var request = new ReqEliminaNota
```

```
        {
            IdOperazioneClient = Guid.NewGuid(),
            IdNota = entita.IdServer.Value
        };

        await CodaOfflineService.AccodaAsync(
            tipoOperazione: NotaOffline.Delete,
            endpoint: "api/Note/Elimina",
            payload: request,
            storeNameEntitaLocale: StoreName,
            keyEntitaLocale: GetKey(entita.Id),
            azioneCoda: AzioniCodaOffline.Delete,
            cancellationToken: cancellationToken);
    }
}
```

In questo esempio, entrambi gli endpoint accettano richieste **HTTP POST**, coerentemente con il comportamento attuale di `AccodaAsync`.

Per eliminare una nota si utilizza il metodo ereditato:

```
await notaService.EliminaAsync(idNota, cancellationToken);
```

La classe base distingue automaticamente tra nota ancora locale e nota già sincronizzata.

> □□ **L’esempio mantiene stabile l’ID locale**
> Dopo la creazione sul server, `Id` resta invariato e `IdServer` viene valorizzato con l’identificativo ricevuto. In questo modo il collegamento tra UI, record locale e coda rimane stabile.

4. Gestisci la conferma del server

L’handler aggiorna il record locale dopo una creazione riuscita e lo rimuove dopo un’eliminazione confermata.

L’esempio assume che l’endpoint di creazione restituisca direttamente un oggetto `NotaResponse`.

```
public sealed class NotaSyncSuccessHandler : ISyncSuccessHandler
{
    private readonly IIndexedDbLocalDataStoreService _localDataStore;
```

```
    public NotaSyncSuccessHandler(IIndexedDbLocalDataStoreService localDataStore)
    {
        _localDataStore = localDataStore;
    }

    public bool CanHandle(string tipoOperazione)
    {
        return tipoOperazione == NotaOffline.Create || tipoOperazione == NotaOffline.Delete;
    }

    public async Task GestisciSuccessoAsync( CustomOggettoInCoda item, string? responseJson,
CancellationToken cancellationToken = default)
    {
        string store = item.StoreNameEntitaLocale ?? throw new
InvalidOperationException("Store locale mancante.");

        string key = item.KeyEntitaLocale ?? throw new InvalidOperationException("Chiave
locale mancante.");

        if (item.TipoOperazione == NotaOffline.Delete)
        {
            await _localDataStore.EliminaAsync( store, key, cancellationToken);

            return;
        }

        var risposta = JsonSerializer.Deserialize<NotaResponse>( responseJson ?? throw new
InvalidOperationException("Risposta vuota."),
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        if (risposta is null || risposta.Id == Guid.Empty)
        {
            throw new InvalidOperationException( "La risposta non contiene un identificativo
valido.");
        }
```

```
        var nota = await _localDataStore.RicercaByKeyAsync<CustomNotaLocale>( store, key,
cancellationToken);

        if (nota is null)
        {
            throw new InvalidOperationException( "La nota locale da sincronizzare non è
disponibile.");
        }

        nota.IdServer = risposta.Id;
        nota.Testo = risposta.Testo;
        nota.StatoSyncLocale = EStatoSyncLocale.Sincronizzata;

        await _localDataStore.SalvaAsync( store, key, nota, cancellationToken);
    }
}
```

Non è necessario chiamare `SetAsSincronizzatoAsync` dentro l'handler: lo esegue `SyncService` dopo che gli handler sono terminati correttamente.

> ⚠ **Gestisci esplicitamente le anomalie**
> Se il server ha creato la nota ma l'handler fallisce, l'operazione può essere ritentata. L'API deve riconoscere lo stesso `IdOperazioneClient` e restituire il risultato già prodotto senza creare duplicati.

5. Registra servizio e handler

Aggiungi le registrazioni specifiche dell'app:

```
services.AddScoped<NotaService>();

services.AddScoped<ISyncSuccessHandler, NotaSyncSuccessHandler>();
```

I servizi condivisi di storage, coda, connettività e sincronizzazione devono essere già registrati tramite la configurazione Core del progetto.

6. Utilizza il servizio dalla pagina

La pagina lavora con il servizio applicativo e rilegge l'elenco locale dopo le operazioni.

```
private List<CustomNotaLocale> Note = new();

private async Task CreaNotaAsync(string testo)
{
    await NotaService.CreaAsync(testo);

    Note = await NotaService.ElencoLocaleAsync();

    // Mostrare una conferma di salvataggio locale.
}

private async Task EliminaNotaAsync(Guid id)
{
    await NotaService.EliminaAsync(id);

    Note = await NotaService.ElencoLocaleAsync();
}
```

Per un comando manuale di sincronizzazione:

```
private async Task SincronizzaAsync()
{
    var esiti = await SyncService.SincronizzaAsync();

    Note = await NotaService.ElencoLocaleAsync();

    // Esaminare gli esiti e gli eventuali elementi ancora pendenti.
}
```

Quando è attivo `SyncMonitorService`, la sincronizzazione può partire anche automaticamente. La pagina deve quindi aggiornarsi agli eventi appropriati, come `SyncCompletata` e `SyncFallita`, rimuovendo le sottoscrizioni alla propria chiusura.

7. Verifica i casi principali

Prova	Risultato atteso
Crea una nota offline	Record locale con stato `DaCreare` e operazione `Create` pendente.
Elimina la nota prima della sincronizzazione	Rimozione del record locale e delle operazioni pendenti collegate.

Prova	Risultato atteso
Sincronizza una creazione	`IdServer` valorizzato, stato `Sincronizzata` e operazione rimossa dalla coda.
Elimina una nota sincronizzata	Nota nascosta localmente e operazione `Delete` pendente.
Sincronizza l’eliminazione	Rimozione definitiva del record locale.
Simula un errore API	Operazione ancora pendente, errore memorizzato e contatore incrementato.

> **□□ Prima dell’utilizzo in produzione**
> Completa l’esempio con la gestione del fallimento tra salvataggio e accodamento, il coordinamento delle eliminazioni mentre una sincronizzazione è in corso e l’isolamento dei dati per utente e azienda. I tre servizi non risolvono automaticamente questi casi.

Certificati HTTPS self-signed per IIS compatibili con Android e dispositivi Apple

Questa procedura permette di pubblicare un sito IIS interno in HTTPS utilizzando un indirizzo IP privato, ad esempio:

```
https://192.168.232.122
```

La soluzione utilizza:

```
Soluziona Local CA
        ↓ firma
Certificato server IIS
        ↓
https://IP_SERVER
```

Sul tablet o smartphone viene installata solamente la CA `Soluziona Local CA`.

Il certificato server rimane installato sul server IIS.

Questa configurazione è adatta a WebApp interne, comprese PWA e Blazor WebAssembly, che devono essere considerate HTTPS valide dai browser dei dispositivi aziendali.

---

1. Requisiti

Il server deve avere:

- Windows Server / Windows con IIS;
- PowerShell eseguito come amministratore;
- indirizzo IP locale statico;
- porta HTTPS raggiungibile dai dispositivi della LAN.

Esempio utilizzato nella guida:

```
192.168.232.122
```

Sostituire questo IP con quello effettivo del server.

È importante che l'IP sia statico, perché il certificato server sarà valido specificamente per quell'indirizzo.

---

2. Creare la cartella dei certificati

Aprire **PowerShell come amministratore** ed eseguire:

```
New-Item -ItemType Directory -Path C:\CertificatiSoluziona -Force
```

---

3. Creare la CA locale

La CA locale è il certificato root che verrà considerato attendibile dai dispositivi aziendali.

Eseguire:

```
$ca = New-SelfSignedCertificate -Type Custom -Subject "CN=Soluziona Local CA" -FriendlyName "Soluziona Local CA" -CertStoreLocation "Cert:\LocalMachine\My" -KeyAlgorithm RSA -KeyLength 2048 -HashAlgorithm SHA256 -KeyExportPolicy Exportable -KeyUsage CertSign,CRLSign -TextExtension @("2.5.29.19={critical}{text}CA=true") -NotAfter (Get-Date).AddYears(10)
```

La CA deve avere obbligatoriamente:

```
Basic Constraints
CA = TRUE
```

e:

```
Key Usage
Certificate Signing
CRL Signing
```

Microsoft supporta `CertSign` e `CRLSign` come utilizzi della chiave per certificati CA.

4. Esportare la CA pubblica

Eseguire:

```
Export-Certificate -Cert $ca -FilePath "C:\CertificatiSoluziona\SoluzionaLocalCA.cer"
```

Verrà creato:

```
C:\CertificatiSoluziona\SoluzionaLocalCA.cer
```

Questo è il file da distribuire sui tablet e smartphone.

Non deve essere esportata né distribuita la chiave privata della CA.

---

5. Installare la CA anche sul server Windows

Per fare in modo che il server consideri attendibile la propria CA:

```
Import-Certificate -FilePath "C:\CertificatiSoluziona\SoluzionaLocalCA.cer" -CertStoreLocation
"Cert:\LocalMachine\Root"
```

---

6. Verificare che la CA sia corretta

Eseguire:

```
certutil -dump C:\CertificatiSoluziona\SoluzionaLocalCA.cer
```

Controllare che siano presenti:

```
Basic Constraints
CA=TRUE
```

e:

```
Key Usage
Certificate Signing
CRL Signing
```

Se invece compaiono solamente:

```
Digital Signature
Key Encipherment
```

il certificato non è stato creato correttamente come CA e non deve essere distribuito ai dispositivi.

---

7. Creare il certificato HTTPS del server

Impostare prima l'IP:

```
$ip = "192.168.232.122"
```

Poi creare il certificato server firmato dalla CA:

```
$cert = New-SelfSignedCertificate -Type Custom -Subject "CN=$ip" -FriendlyName "Soluziona WASM - $ip" -Signer $ca -CertStoreLocation "Cert:\LocalMachine\My" -KeyAlgorithm RSA -KeyLength 2048 -HashAlgorithm SHA256 -KeyExportPolicy Exportable -KeyUsage DigitalSignature,KeyEncipherment -TextExtension @("2.5.29.17={text}IPAddress=$ip","2.5.29.37={text}1.3.6.1.5.5.7.3.1") -NotAfter (Get-Date).AddYears(1)
```

Il certificato contiene:

```
Subject:
CN=192.168.232.122
```

e soprattutto:

```
Subject Alternative Name:
IP Address = 192.168.232.122
```

L'IP deve essere presente nel **Subject Alternative Name (SAN)**. Non è sufficiente inserirlo solamente nel Common Name.

Microsoft documenta esplicitamente l'utilizzo di:

```
2.5.29.17={text}IPAddress=...
```

per inserire un indirizzo IP nel SAN.

L'altra estensione:

```
1.3.6.1.5.5.7.3.1
```

corrisponde a:

```
Server Authentication
```

necessario per un certificato TLS server.

---

8. Verificare il certificato server

Eseguire:

```
$cert | Format-List Subject,Issuer,Thumbprint,NotAfter
```

Il risultato deve essere simile a:

```
Subject    : CN=192.168.232.122
Issuer     : CN=Soluziona Local CA
Thumbprint : ...
NotAfter   : ...
```

Quindi:

```
Rilasciato a:
192.168.232.122

Rilasciato da:
Soluziona Local CA
```

`Rilasciato a` e `Rilasciato da` non devono essere uguali.

---

9. Configurare IIS

Aprire:

```
IIS Manager
→ Sites
→ sito Web
→ Bindings...
```

Se esiste già il binding HTTPS:

```
https
porta 443
```

selezionarlo e premere:

```
Edit...
```

Nel campo:

```
SSL certificate
```

selezionare:

```
Soluziona WASM - 192.168.232.122
```

Non selezionare:

```
Soluziona Local CA
```

La CA non deve essere utilizzata direttamente come certificato HTTPS del sito.

La configurazione corretta è:

```
IIS
└── Soluziona WASM - 192.168.232.122
      └── firmato da Soluziona Local CA
```

Facoltativamente:

```
iisreset
```

normalmente non è necessario, ma può essere utilizzato per riavviare IIS dopo la modifica.

---

10. Configurazione finale dei certificati

Sul server IIS:

```
Soluziona Local CA
        ↓
firma
        ↓
Soluziona WASM - 192.168.232.122
        ↓
IIS HTTPS
```

Sul dispositivo:

```
Soluziona Local CA
```

Il certificato `Soluziona WASM - 192.168.232.122` non deve essere installato sul tablet.

---

11. Installazione su Android

Copiare sul dispositivo:

```
SoluzionaLocalCA.cer
```

È possibile utilizzare:

- Quick Share;
- USB;
- file server interno;
- altro sistema aziendale di trasferimento file.

Non aprire semplicemente il `.cer` dal file manager scegliendo un'installazione generica: Android potrebbe tentare di installarlo come certificato utente/client e chiedere una chiave privata.

Aprire invece le **Impostazioni Android**.

Il percorso varia in base al produttore, ma generalmente è simile a:

```
Impostazioni
→ Sicurezza e privacy
→ Altre impostazioni di sicurezza
→ Crittografia e credenziali
→ Installa un certificato
→ Certificato CA
```

Selezionare:

```
SoluzionaLocalCA.cer
```

Android potrebbe visualizzare un avviso relativo all'installazione manuale di una CA. È normale.

Dopo l'installazione verificare:

```
Impostazioni
→ Sicurezza
→ Credenziali attendibili
→ Utente
```

Deve comparire:

```
Soluziona Local CA
```

---

12. Verifica su Android

Collegare il tablet alla rete aziendale e aprire Chrome:

```
https://192.168.232.122
```

Il sito deve aprirsi senza:

```
NET::ERR_CERT_AUTHORITY_INVALID
```

e senza:

```
La connessione non è privata
```

Controllando le informazioni del sito, la connessione deve risultare sicura.

Se il sito è una PWA, a questo punto Chrome può considerarlo un **secure context**, requisito necessario per funzionalità quali Service Worker e installazione PWA.

---

13. Installazione su iPhone e iPad

Copiare sul dispositivo Apple:

```
SoluzionaLocalCA.cer
```

È possibile trasferirlo tramite:

- AirDrop;
- e-mail aziendale;
- file server;
- MDM;
- Apple Configurator.

Aprire il certificato sul dispositivo.

iOS/iPadOS propone l'installazione di un profilo.

Procedere da:

```
Impostazioni
→ Generali
→ VPN e gestione dispositivo
```

oppure dalla voce:

```
Profilo scaricato
```

e completare l'installazione.

---

14. Abilitare l'attendibilità completa su Apple

Questo passaggio è fondamentale.

Un certificato root installato manualmente su iPhone/iPad **non viene automaticamente considerato attendibile per SSL/TLS**.

Apple richiede di andare in:

```
Impostazioni
→ Generali
→ Info
→ Attendibilità certificati
```

Sotto:

```
Abilita attendibilità completa per certificati root
```

attivare:

```
Soluziona Local CA
```

e confermare.

Apple documenta esplicitamente questo passaggio per i certificati root installati manualmente.

Se `Soluziona Local CA` non compare in questa schermata, il certificato root non è stato installato correttamente.

---

15. Verifica su iPhone/iPad

Aprire Safari:

```
https://192.168.232.122
```

Il sito deve aprirsi senza avvisi relativi al certificato.

Il certificato server utilizzato da IIS deve rispettare almeno:

```
RSA >= 2048 bit
SHA-256 / SHA-2
Extended Key Usage = Server Authentication
Subject Alternative Name corretto
```

Apple richiede RSA da almeno 2048 bit, SHA-2 e `serverAuth` per i certificati TLS server moderni.

Nel nostro certificato questi requisiti vengono impostati durante la generazione.

---

16. Durata dei certificati

Configurazione consigliata:

```
Soluziona Local CA:
10 anni

Certificato server:
1 anno
```

Quando scade il certificato server non è necessario reinstallare la CA sui tablet.

È sufficiente generare un nuovo certificato:

```
Soluziona WASM - IP
```

firmato dalla stessa:

```
Soluziona Local CA
```

e sostituirlo nel binding HTTPS di IIS.

I dispositivi continueranno a considerarlo attendibile perché possiedono già la CA root.

Apple applica limiti specifici alla durata dei certificati pubblicamente attendibili; il limite dei 398 giorni non si applica alle root aggiunte manualmente dall'utente o dall'amministratore.

Per uniformità e manutenzione è comunque consigliabile rinnovare il certificato server annualmente.

---

17. Aggiungere altri server o applicazioni

La stessa `Soluziona Local CA` può firmare più certificati.

Esempio:

```
Soluziona Local CA
├── 192.168.232.122
├── 192.168.232.123
├── 192.168.232.150
└── altri server interni
```

La CA deve essere installata una sola volta sui dispositivi.

Per ogni nuovo server è sufficiente generare un certificato specifico per il relativo IP.

Esempio:

```
$ip = "192.168.232.123"

$cert = New-SelfSignedCertificate -Type Custom -Subject "CN=$ip" -FriendlyName "Soluziona
HTTPS - $ip" -Signer $ca -CertStoreLocation "Cert:\LocalMachine\My" -KeyAlgorithm RSA -
KeyLength 2048 -HashAlgorithm SHA256 -KeyExportPolicy Exportable -KeyUsage
DigitalSignature,KeyEncipherment -TextExtension
```

```
@("2.5.29.17={text}IPAddress=$ip","2.5.29.37={text}1.3.6.1.5.5.7.3.1") -NotAfter (Get-Date).AddYears(1)
```

---

18. Attenzione alla chiave privata della CA

La chiave privata della:

```
Soluziona Local CA
```

permette di creare certificati considerati attendibili da tutti i dispositivi sui quali è stata installata.

Deve quindi essere protetta.

È consigliato effettuare un backup protetto da password della CA:

```
$password = Read-Host "Password backup CA" -AsSecureString
Export-PfxCertificate -Cert $ca -FilePath "C:\CertificatiSoluziona\SoluzionaLocalCA-PRIVATE.pfx" -Password $password
```

Il file:

```
SoluzionaLocalCA-PRIVATE.pfx
```

deve essere conservato in un luogo sicuro e **non deve essere distribuito sui tablet**.

Ai dispositivi deve essere fornito solamente:

```
SoluzionaLocalCA.cer
```

---

19. Differenza tra `.CER` e `.PFX`

`.CER`

Contiene la parte pubblica del certificato.

Esempio:

```
SoluzionaLocalCA.cer
```

Può essere distribuito ai dispositivi.

Non contiene la chiave privata.

.PFX

Può contenere:

```
certificato
+
chiave privata
```

Deve essere protetto con password e conservato esclusivamente dagli amministratori.

Non distribuire il PFX della CA ai tablet.

---

20. Problemi comuni

Android dice "L'installazione di un certificato richiede la chiave privata"

Il certificato è stato aperto con il flusso di installazione per certificati client.

Usare invece:

```
Impostazioni
→ Installa certificato
→ Certificato CA
```

Il certificato non compare nelle credenziali attendibili Android

Controllare:

```
certutil -dump C:\CertificatiSoluziona\SoluzionaLocalCA.cer
```

Deve contenere:

```
CA=TRUE
```

e:

```
Certificate Signing
CRL Signing
```

Chrome continua a segnalare certificato non valido

Verificare che IIS utilizzi:

```
Soluziona WASM - IP
```

e non:

```
Soluziona Local CA
```

Verificare inoltre che il SAN contenga esattamente l'IP utilizzato dal dispositivo.

Esempio:

```
URL:
https://192.168.232.122
```

deve corrispondere a:

```
Subject Alternative Name:
IP Address = 192.168.232.122
```

Il certificato funziona sul server ma non sul tablet

È normale se la CA non è stata installata sul tablet.

Il server può già considerare attendibile la CA attraverso il proprio archivio certificati, mentre Android/iOS possiedono archivi separati.

---

iPhone/iPad ha installato il certificato ma Safari continua a segnalarlo

Controllare:

```
Impostazioni
→ Generali
→ Info
→ Attendibilità certificati
```

e abilitare:

```
Attendibilità completa
→ Soluziona Local CA
```

Apple specifica che i certificati root installati manualmente non sono automaticamente attendibili per SSL/TLS.

---

21. Schema finale

```
WINDOWS SERVER / IIS


Soluziona Local CA
      │
      │ firma
      ▼
Soluziona WASM - 192.168.232.122
      │
      ▼
IIS :443
      │
      ▼
https://192.168.232.122
```

Dispositivi:

```
ANDROID
└── Soluziona Local CA
    installata come Certificato CA

iPHONE / iPAD
└── Soluziona Local CA
    installata
    +
    Attendibilità completa abilitata
```

IIS:

```
Binding HTTPS
└── Soluziona WASM - 192.168.232.122
```

Non utilizzare direttamente `Soluziona Local CA` nel binding HTTPS di IIS.