# Struttura di un'applicazione

## Indice generale

# Struttura di un'applicazione

Scopri l’architettura degli oggetti di un’applicazione Instant Developer Cloud

## Introduzione

Nel capitolo Introduzione a Instant Developer Cloud abbiamo visto come un progetto sia costituito da applicazioni, data model e librerie. Abbiamo inoltre descritto i vari container in cui ogni applicazione può funzionare: il container per applicazioni web basato su Node.js, quello per applicazioni mobile basato su Apache Cordova ed infine le PWA.

In questo capitolo entreremo all’interno di un’applicazione per studiare l’architettura degli oggetti.

Per una migliore comprensione del testo seguente vi ricordiamo che la programmazione di Instant Developer Cloud è basata su JavaScript e che il modello di Object Orientation utilizzato è basato sull’uso di *prototype*. Per il lettore che non avesse familiarità con questi concetti si consiglia un approfondimento preventivo.

### Applicazione e sessione

L’oggetto base dell'architettura di un’applicazione Instant Developer Cloud è un oggetto messo a disposizione dal framework denominato *App*. Esso è un oggetto di infrastruttura, cioè contiene la definizione di tutte le classi di codice necessarie al funzionamento dell’applicazione. I tipi principali di classi contenuti in *App* sono i seguenti:

- Classi del framework: ad esempio *App.View* è la classe base di tutte le videate.
- Classi o videate specifiche dell’applicazione.
- Librerie definite nel progetto.

*App* è un oggetto definito dal framework al momento dell’attivazione di un processo worker del container in cui l’applicazione è in funzione. In particolare:

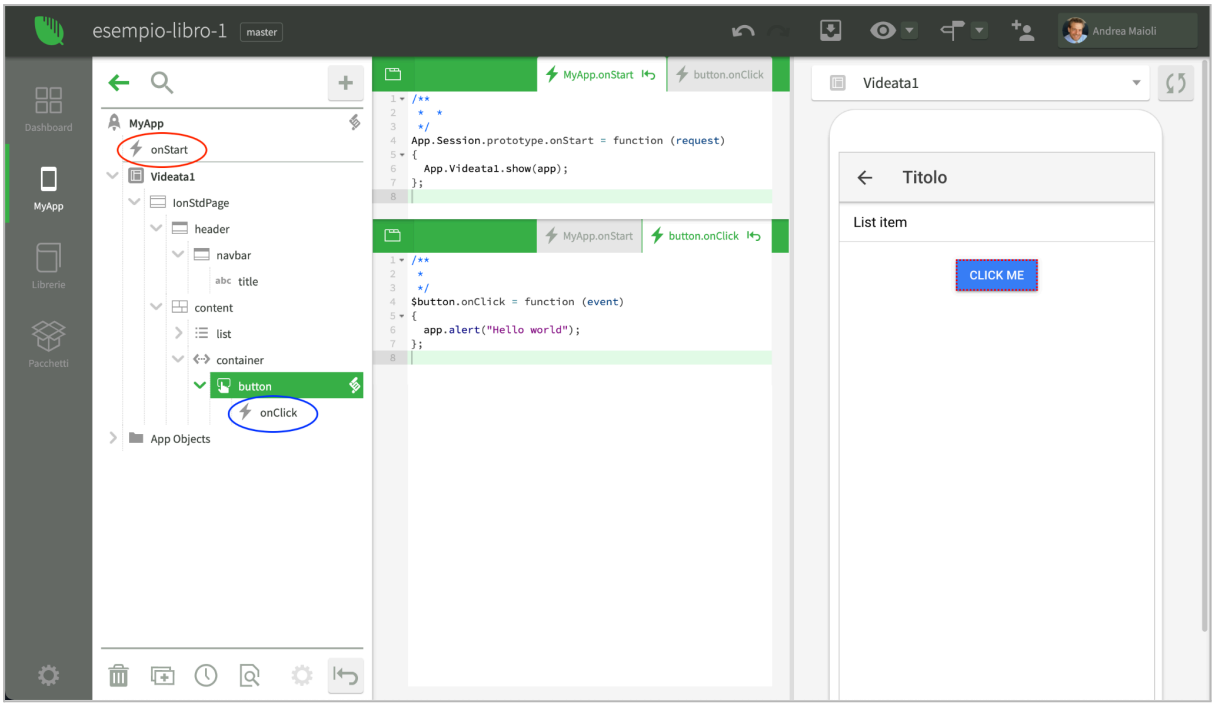
- Per il container di applicazioni web: all’avvio di un nuovo worker all’interno di uno specifico processo Node.js, vengono istanziati l’oggetto *App* e tutte le sue definizioni. Nel medesimo container possono essere presenti più oggetti *App* separati, uno per ogni processo worker del container.
- Per il container di applicazioni mobile e PWA: esso supporta un unico processo in cui è in esecuzione l’unica sessione applicativa del device o browser; in questo caso verrà creato un solo oggetto *App* per l’intero container.

*App* gestisce la lista delle sessioni applicative associate al processo in cui esso è presente. Ogni sessione applicativa è un’istanza della classe *Session* del framework ed è referenziabile in tutto il codice dell’applicazione tramite la variabile *app*. La sessione è un oggetto di lavoro, quindi conterrà le istanze delle classi che compongono l’applicazione, man mano che la sessione ne richiede l’utilizzo.

Sono presenti vari tipi di sessione; la più comune è quella collegata ad un browser di un utente che si collega al container e vuole iniziare a lavorare con l'applicazione web.

Ricapitolando: l'oggetto *App* (con la A maiuscola) rappresenta l'infrastruttura, quindi contiene la definizione delle classi dell'applicazione. L'oggetto *app* (con la a minuscola) rappresenta la sessione di lavoro, quindi contiene le istanze delle classi dell'applicazione che la sessione di lavoro sta utilizzando per svolgere i suoi compiti.

Nell'immagine sottostante è mostrato un progetto di base che esemplifica questi concetti.




Nell'applicazione chiamata *MyApp* è contenuta una videata chiamata *Videata1*, mostrata in anteprima sulla destra. Al centro vediamo il codice dell'evento *onStart*, evidenziato in rosso, e il codice dell'evento *onClick* del pulsante *Click Me* evidenziato in blu.

L'evento *onStart* viene chiamato dal framework al momento dell'attivazione di una nuova sessione e l'espressione usata per aprire un'istanza della videata è la seguente:

```
App.Videata1.show(app);
```

Quindi all'avvio di una sessione viene invocato il metodo statico *show* della classe *Videata1* che, come indicato in precedenza, viene definita all'interno dell'oggetto *App* che rappresenta l'intera infrastruttura delle classi dell'applicazione *MyApp*.

Come avviene per tutte le invocazioni di metodi statici in Instant Developer Cloud, come primo parametro viene sempre passato *app*, cioè il riferimento alla sessione di lavoro per cui il metodo viene invocato. Ritornando all'esempio del metodo *show*, è solo in questo modo che è possibile per la classe che gestisce la videata apparire nel browser giusto, cioè quello collegato alla sessione di lavoro appena iniziata.

Il metodo *show*, invocato in modo statico sulla classe della videata, agisce come segue:

1) Crea un'istanza della classe - cioè l'oggetto videata vero e proprio - a cui passa il riferimento alla sessione in cui la videata viene aperta.
2) Memorizza l'oggetto videata appena creato in una lista di videate aperte all'interno della sessione (*app*).
3) Chiama un metodo interno dell'oggetto videata che inizializza tutti gli elementi visuali contenuti in essa.
4) L'oggetto videata comunica alla sessione di creare nel browser i corrispondenti elementi visuali utilizzando un metodo del motore di rendering grafico che è parte del framework.

Vediamo adesso il codice dell'evento *onClick* relativo al pulsante *Click Me* presente nella videata. Il codice apre una message box nel browser mostrando il testo "Hello world". Per ottenere questo risultato l'espressione utilizzata è la seguente:

```
app.alert("Hello World");
```

In questo caso viene invocato il metodo *alert* sull'oggetto *app* che rappresenta la sessione collegata al browser in cui è stato cliccato il pulsante. Il metodo *alert* è naturalmente presente nell'oggetto *app* (sessione) in quanto il suo scopo è proprio quello di mostrare un messaggio nel browser collegato alla sessione.

# Tipi di sessione

Il container per applicazioni web supporta i seguenti tipi di sessione:

1. *web*: normali sessioni web che vengono istanziate tramite browser.
2. *rest*: sessioni che gestiscono un comando in modalità *rest*, solitamente proveniente da un sistema esterno che deve essere integrato.
3. *webapi*: sessioni che gestiscono una webapi di tipo OData generata con Instant Developer Cloud.
4. *proxy*: sessioni che permettono ai dispositivi mobili di collegarsi con il backend del cloud per condividere dati o sincronizzare database.
5. *server*: sessioni batch che svolgono attività periodiche in modalità non presieduta.

Il container per applicazioni mobile e PWA supporta solo una singola sessione di tipo *web*.

## Avvertenza

Nel container per applicazioni web, tramite *App* è possibile condividere informazioni tra sessioni o addirittura è possibile per una sessione operare all'interno di un'altra. Questo comportamento è sconsigliato perché può causare effetti collaterali difficili da identificare. In ogni caso non consentirebbe di raggiungere l'intero insieme delle sessioni in esecuzione, che viene suddiviso fra più processi Node.js e quindi in oggetti *App* non presenti nella medesima virtual machine JavaScript.
Per ottenere questo scopo è invece possibile utilizzare il framework di sincronizzazione, in grado di comunicare in modo sicuro a tutte le sessioni in esecuzione in un determinato container per applicazioni web.

# Ciclo di vita di una sessione

Le sessioni iniziano e terminano in modo diverso a seconda del loro tipo, e a seconda di questi diversi modi, notificano eventi diversi.

## Sessioni web

La sessione nasce quando un browser contatta il server Node.js e, dopo l'attivazione del framework lato client, viene attivata la connessione websocket. In questo momento nel codice dell'applicazione viene notificato l'evento *onStart* in cui è possibile leggere la richiesta originale del browser.

A differenza di un'applicazione web tradizionale, le applicazioni sviluppate con Instant Developer Cloud non hanno tempo di attesa prestabilito per una sessione inattiva (solitamente 20 minuti). La sessione rimane attiva fino a che il browser viene chiuso, a meno di non impostare la proprietà *app.sessionTimeout* che permette di terminare la sessione dopo un determinato periodo di inattività.

Quando il browser viene chiuso, la sessione viene terminata immediatamente in quanto il framework client comunica subito al server che è in fase di chiusura. In alcune situazioni la connessione al server si interrompe senza poter dare notizia della chiusura del browser; in questo caso, dopo un periodo di tempo pari a 3 secondi, la sessione viene terminata lato server. Quando la sessione viene terminata, viene notificato l'evento *onTerminate* a livello di sessione; una sessione può essere terminata anche dal codice dell'applicazione chiamando il metodo *app.terminate()*.

Se il browser entra in background, notifica all'applicazione l'evento *onPause*. In questo stato, quando ritorna in primo piano, verrà notificato l'evento *onResume*.

Le sessioni web, infine, notificano l'evento *onHistoryPop* quando l'utente clicca sul pulsante *back* del browser, fino alla prima pagina di ingresso nell'applicazione.

## Sessioni REST (webapi)

Le sessioni REST vengono attivate quando un agente (browser o altro) contatta il server inserendo il parametro *mode=rest* nella query string. In questo caso la sessione ha un ciclo di vita completamente diverso dal precedente, infatti al posto dell'evento *onStart* viene notificato l'evento *onCommand*, nel quale è possibile leggere tutti i parametri della richiesta compresi eventuali file allegati alla medesima. I file vengono gestiti dal framework prima di attivare l'evento *onCommand*, cioè vengono salvati sul file system dell'applicazione nella cartella *uploaded*.

La sessione REST può rispondere all'agente chiamante tramite il metodo *app.sendResponse*. Dopo questa chiamata la sessione viene terminata, quindi la risposta deve essere inviata in una volta sola. Il timeout di risposta è impostato a 60 secondi, ed è modificabile tramite la proprietà *app.sessionTimeout.* Se la risposta non viene data in tempo, la sessione viene terminata con errore.

Esistono due casi particolari di sessioni che si attivano in modalità REST pur non avendo la stringa *mode=rest* nella query string. Il primo riguarda una chiamata ad una webapi di tipo *OData* generata con Instant Developer Cloud. Per distinguere questo caso dalle normali chiamate REST è possibile utilizzare il metodo *app.isWebApiRequest()* che restituisce *true* se la chiamata è di tipo webapi OData.

Il secondo caso si verifica quando la chiamata all'applicazione avviene da parte di un crawler bot, come ad esempio quello di Google. In questo caso la chiamata viene comunque attivata in modalità REST, così da poter restituire al motore di ricerca una stringa HTML che rappresenta il contenuto della pagina richiesta. Diventa così possibile indicizzare nei motori di ricerca le applicazioni sviluppate con Instant Developer Cloud in modalità SPA (single page application), particolarmente difficili da far indicizzare a tali motori.

## Esempio di indicizzazione sui motori di ricerca

L'indicizzazione sui motori di ricerca di una SPA (single page application) è particolarmente difficile perché i motori di ricerca considerano i siti web come una serie di documenti testuali, separati in varie pagine. Una SPA invece è costituita da una sola pagina, quindi i bot dei motori di ricerca non sono in grado di navigare tali pagine come i siti web tradizionali.

Normalmente un'applicazione non necessita di indicizzare ogni pagina, ma solo i contenuti pubblici, cioè quelli che devono essere trovati da chi utilizza i motori di ricerca. Se, ad esempio, pensiamo ad Amazon, nei motori di ricerca troviamo le pagine di dettaglio dei prodotti venduti.

Per ottenere un comportamento analogo, occorre predisporre due template di pagina da gestire per ogni categoria di contenuto pubblico. Se, ad esempio, vogliamo indicizzare un elenco di case in vendita, potremo gestire i seguenti tipi di query string:

1) https://myapp.com/?cat=case
2) https://myapp.com/?cat=case&cid=<id della casa>

All'interno dell'evento *onStart* dell'applicazione che viene notificato da un vero browser, nel primo caso occorre mostrare la pagina di ricerca delle case, mentre nel secondo caso la pagina di dettaglio della casa identificata dal suo *id*.

Per ottenere l'indicizzazione occorre gestire anche l'evento *onCommand*, che viene notificato per le medesime query string da un crawler di un motore di ricerca. Nel primo caso occorre restituire una stringa HTML composta dalla lista di case contenute nel database, eventualmente suddividendola in più pagine se la lista contiene migliaia di referenze. Per ogni riga della lista occorre inserire un link del secondo tipo, in modo che il motore di ricerca entri in ogni pagina di dettaglio. Se l'evento *onCommand* viene chiamato con un link del secondo tipo, sarà sufficiente restituire una stringa HTML che rappresenta il contenuto semantico delle informazioni riguardanti il dettaglio della casa.

In questo modo si otterrà l'indicizzazione della lista e di ogni dettaglio. Quando un browser cliccherà sul link di un motore di ricerca, verrà richiamata la nostra applicazione in modo da creare una sessione web in cui l'evento *onStart* si preoccupa di visualizzare subito le informazioni trovate tramite il motore di ricerca.

## Sessioni offline

Una sessione offline nasce quando un utente attiva l'applicazione sul proprio device o workstation. Le applicazioni offline vengono installate come app scaricabili da app store per gli smartphone e tablet, oppure come PWA multi-channel. Il termine offline non identifica la mancanza di connessione internet, ma il fatto che l'intera applicazione è in funzione localmente nel dispositivo e continuerà a funzionare anche in mancanza di connessione internet.

La sessione offline si comporta a tutti gli effetti come una sessione web. Occorre però tenere in considerazione le differenze nell'ambiente operativo in cui essa è in esecuzione, fra le quali:

- La sessione web è in funzione in un server Node.js, la sessione offline in una virtual machine di un device o di un browser (PWA). In questo caso la virtual machine dell'applicazione (framework server) è la stessa di quella del motore di rendering (framework client).
- Il database utilizzabile da una sessione web è di tipo Postgres, o comunque è un database cloud. Una sessione offline può utilizzare SQLite; alcuni tipi di PWA non hanno accesso a nessun tipo di database offline. Per maggiori informazioni sulla gestione dei database, si consiglia di leggere il capitolo relativo.
- Il file system dell'applicazione nel cloud ha caratteristiche diverse da quello offline anche se l'interfaccia di programmazione è la stessa. Le PWA possono accedere ai file offline solo in alcuni casi.
- Una sessione offline può accedere ai plugin nativi del device; una sessione web o PWA presenta limitazioni maggiori.

Per distinguere se la sessione è di tipo web o di tipo offline è possibile utilizzare il metodo *app.runsLocally()* che restituisce *true* se la sessione è di tipo offline.

## Sessioni proxy

Una sessione proxy rappresenta la controparte cloud di una sessione offline quando l'applicazione utilizza il framework di sincronizzazione incluso in Instant Developer Cloud. Tale framework permette di implementare efficacemente un'architettura di tipo client-cloud senza dover modificare il codice dell'applicazione scritto per la versione web della medesima.

La sessione proxy nasce quando una sessione offline attiva il sistema di sincronizzazione ed in questo momento viene notificato alla sessione proxy l'evento *onConnect*, che permette di controllare l'accesso e gestire l'invio dei dati al database offline. Quando la sessione offline si disconnette, alla sessione proxy viene notificato l'evento *onDisconnect* e poi la sessione viene terminata.

Per maggiori informazioni sulle sessioni proxy si consiglia di leggere il capitolo relativo al sistema di sincronizzazione.

## Sessioni server

Una sessione server è in esecuzione nel server Node.js senza avere una controparte browser; di solito viene utilizzata per eseguire processi in modalità batch. Una sessione server funziona a tutti gli effetti come una sessione web, ma senza visualizzare l'interfaccia utente della medesima.

La sessione server nasce automaticamente al momento dell'installazione dell'applicazione su un server se per questa applicazione è stato attivato il flag *Attiva server session* all'interno della console.

Solitamente nell'evento *onStart* viene utilizzato il metodo *app.isServerSession*() per aprire videate specifiche per la gestione dei processi batch della server session. Anche se le videate non vengono visualizzate su un browser utente, il loro comportamento applicativo è identico al caso delle sessioni web.

Per una più semplice gestione delle attività batch, si consiglia di dedicare una videata all'esecuzione di questi processi nella quale sarà presente un timer che permette di eseguire operazioni periodiche.

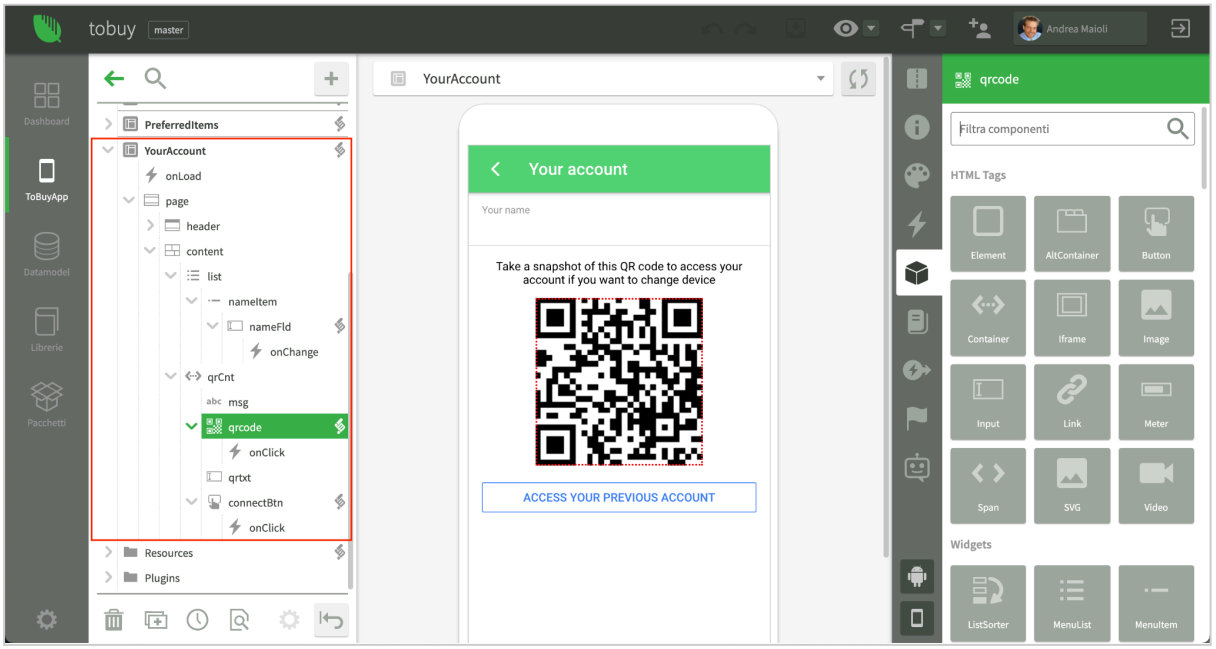
# Le videate

## Definizione di videata

Una videata rappresenta la definizione di una classe di codice di Instant Developer Cloud utilizzata per implementare l'interfaccia utente dell'applicazione. Una classe, per essere considerata come videata, deve derivare, direttamente o indirettamente, dalla classe *Application.View* del framework.

Solitamente le videate vengono inserite all'interno dell'oggetto applicazione di cui fanno parte, ma è possibile definire le videate anche all'interno di librerie di tipo custom. Quest'ultima opzione è particolarmente efficace per videate generiche che possono essere utilizzate in progetti o applicazioni diversi.

All'interno di una videata è possibile inserire elementi grafici di tipo diverso. Tali elementi devono essere presenti nel progetto all'interno della sezione librerie. Il pacchetto *coreLibraries*, incluso in tutti i progetti, contiene la definizione degli elementi base dell'HTML. Di solito si consiglia di aggiungere al progetto il pacchetto *IonicUI* che contiene l'insieme degli elementi grafici del framework Ionic adatto alla creazione di applicazioni di tipo omnichannel.




*La videata YourAccount e i suoi elementi visuali*

Oltre agli elementi visuali, una videata, come ogni altra classe, può includere proprietà, eventi e metodi. Per gli elementi visuali possono essere definiti gli script di gestione degli eventi da essi supportati come si vede per i pulsanti (*onClick*) e per il campo di input (*onChange*).

# Ciclo di vita di una videata

Il ciclo di vita di una videata consiste di soli due stati: una videata può essere aperta oppure chiusa.

## Apertura di una videata

Per aprire una videata, occorre semplicemente chiamare il metodo base *show* su una istanza della stessa oppure in modo statico sulla classe. Nell'esempio precedente, per aprire la videata *YourAccount*, è possibile scrivere uno dei seguenti esempi di codice.

```
App.YourAccount.show(app, options);
```

Oppure:

```
let v = new App.YourAccount(app);
v.show(options);
```

I due metodi non sono equivalenti. Nel primo caso (chiamata statica), il framework crea internamente una *istanza di default* della videata per la sessione app passata come parametro e poi apre quella. Tutti i metodi chiamati in modo statico sulla classe diventano infatti chiamate all'istanza di default per la sessione passata.

Il secondo metodo invece crea una istanza specifica e poi la apre. In questo modo è possibile aprire più volte la stessa videata passando parametri diversi.

Il metodo *show* opera come segue:

1) Crea le istanze degli elementi visuali all'interno dell'istanza in fase di apertura.
2) Notifica l'evento *onLoad* alla videata, senza aspettarne la terminazione se esso è asincrono.
3) Invia al browser tramite il motore di rendering i dati per mostrare effettivamente la videata a video.

Per passare parametri alla videata in fase di apertura occorre utilizzare il parametro *options* del metodo show, come nell'esempio seguente.

```
App.YourAccount.show(app, {account: myacc});
```

I parametri di apertura vengono passati all'evento *onLoad* che li può quindi tenere in considerazione per adeguare la visualizzazione ai dati richiesti. Tra le opzioni che è possibile passare a *show* ce ne sono alcune predefinite che permettono di gestire il modo in cui la videata deve apparire. Ad esempio, inserendo *popup:true*, la videata apparirà sovrapposta alle altre. Dall'IDE è possibile leggere la documentazione completa dei parametri predefiniti.

**Nota importante**: il pacchetto *IonicUI* contiene una videata speciale (*MainPage)* che ha lo scopo di gestire il flusso di navigazione fra videate. Usando *IonicUI* le videate non devono essere aperte direttamente con *show*; si deve invece usare il metodo *push* specifico di *MainPage*. Per maggiori informazioni è possibile leggere la documentazione relativa.

### Chiusura di una videata

La chiusura di una videata avviene chiamando il metodo *close* sull'istanza o in modo statico sulla classe. In tal caso verrà chiusa l'istanza di default della sessione passata.

Di solito una videata viene chiusa da uno script contenuto all'interno della medesima. In tal caso l'istanza della videata in cui lo script è in funzione è accessibile tramite la variabile *view*. Per chiudere la videata attuale è quindi possibile scrivere:

```
view.close(info);
```

L'operazione di chiusura avviene con i seguenti passi:

1) Viene notificato l'evento *onClose* alla videata attuale. Se l'evento restituisce *false*, l'operazione di chiusura viene annullata.
2) Se la videata attuale ha una videata di riferimento (*owner*), viene notificato l'evento *onBack* alla videata di riferimento passando il parametro *info*. La videata di riferimento è quella che tornerà attiva quando quella attuale si chiude.
3) Viene inviato al motore di rendering il comando per eliminare la videata attuale dal browser dell'utente.

**Nota importante**: il pacchetto *IonicUI* contiene una videata speciale (*MainPage)* che ha lo scopo di gestire il flusso di navigazione fra videate. Usando *IonicUI* le videata non devono essere chiuse direttamente con *close*; ma si deve usare il metodo *pop* specifico di *MainPage*. Per maggiori informazioni è possibile leggere la documentazione relativa.

## Gli elementi visuali

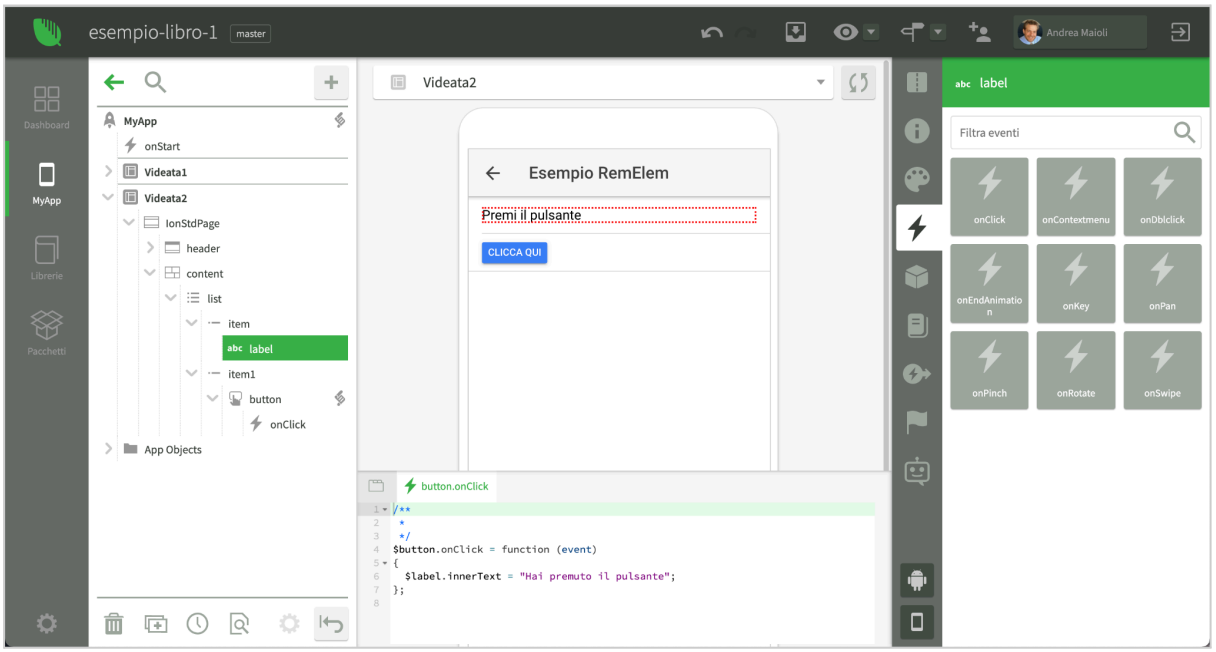
Gli elementi visuali sono i blocchi costitutivi dell'interfaccia utente che una videata mostra nel browser. Essi vengono composti tramite l'IDE, che permette di definire la struttura ad albero degli stessi, come si vede nell'immagine alle pagine precedenti.

Gli elementi disponibili dipendono dalle librerie presenti nel progetto. Il pacchetto *coreLibraries,* incluso in ogni progetto, contiene la definizione degli elementi base dell'HTML oltre ad una serie di elementi di più alto livello come, ad esempio, le mappe, i grafici, gli slider, ecc. Il pacchetto *IonicUI* contiene gli elementi necessari alla creazione di un'applicazione omnichannel responsiva (web e mobile) e il suo utilizzo è altamente consigliato in ogni progetto. È anche possibile aggiungere ulteriori elementi creando un file di interfaccia e definendo una libreria. Per maggiori informazioni fare riferimento al progetto di esempio extensibility-design-pattern.

Tutti gli elementi visuali sono istanze della classe base *App.RemElem* del framework. Ogni istanza viene completata per rappresentare uno specifico tipo di elemento, che può avere metodi o eventi specifici. Ogni *RemElem* si interfaccia automaticamente con il motore di rendering per inviare al browser le modifiche visuali oppure per notificare al codice della videata gli eventi avvenuti nel browser.

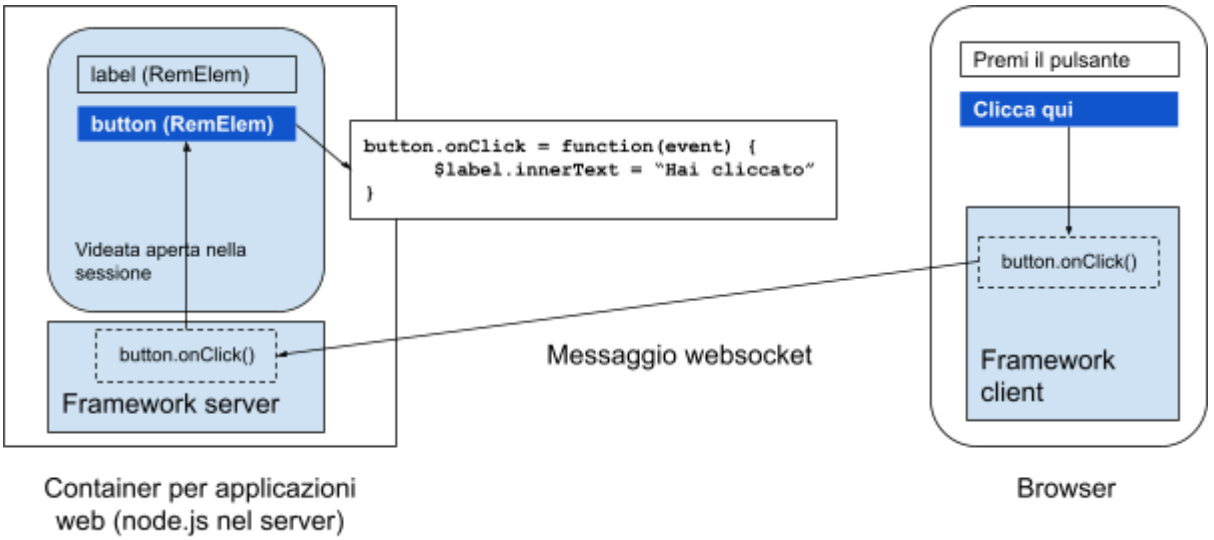
## Esempio di funzionamento

Per comprendere il funzionamento degli elementi visuali e del motore di rendering, si consideri la seguente videata.

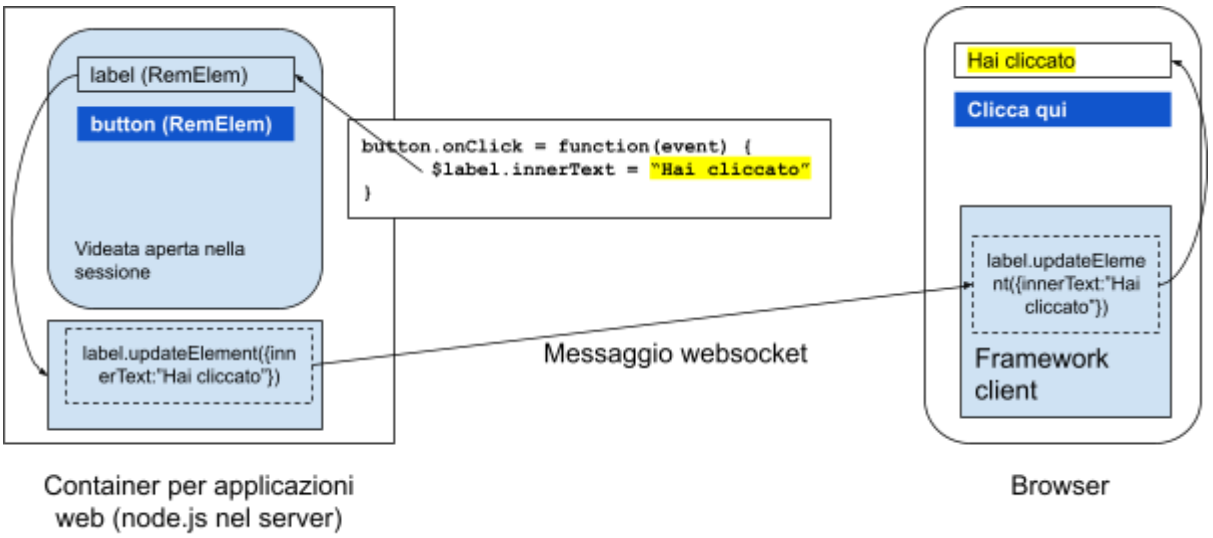



*Cambiare il testo di una label al click*

Quando l'applicazione è in esecuzione e l'utente clicca il pulsante, viene generato un evento *onClick* all'interno del browser, e la parte client del motore di rendering "riflette" tale evento all'istanza di *RemElem* che rappresenta il pulsante nella videata aperta nella sessione in esecuzione nel container per applicazioni web.




Quando il codice dell'evento *onClick* modifica una proprietà dell'elemento *label,* tale modifica viene comunicata al framework client che si occupa di aggiornare il browser. La raccolta delle modifiche avviene attraverso particolari oggetti chiamati Proxy in grado di "sentire" le modifiche applicate ai *RemElem* in modo particolarmente efficiente.




### Confronto fra RemElem e DOM del browser

Gli elementi visuali che compongono una videata sono modellati sulla struttura degli elementi DOM del browser, in particolare supportano un sottoinsieme dell'api di questi ultimi. L'interfaccia di programmazione è descritta nelle librerie di tipo Applicazione contenute nel progetto, in particolare la classe *Application.Element* la cui documentazione è consultabile tramite l'IDE.

Oltre alle proprietà, ai metodi ed agli eventi specifici dei vari tipi di elementi visuali, tutti i RemElem supportano le seguenti caratteristiche:

1) Generazione automatica degli *id*. L'id di ogni elemento viene generato automaticamente e non deve essere modificato.
2) Possibilità di impostare classi CSS tramite la proprietà *className*. Le classi CSS potranno essere manipolate tramite i metodi *addClass*, *removeClass*, *toggleClass* e *containsClass* dell'elemento.
3) Possibilità di impostare stili CSS tramite l'oggetto *style*.
4) Possibilità di impostare attributi dell'elemento o di sotto-oggetti interni all'elemento tramite il metodo *setAttribute*.
5) Manipolazione tramite codice della struttura degli elementi, usando i metodi *clone*, *appendChild*, *insertBefore* e *removeChild.* L'array *elements* contiene tutti gli elementi figli di un determinato *RemElem*.
6) Gestione del focus tramite i metodi *focus* e *blur* e i relativi eventi.

## Contesto di esecuzione del codice di una sessione

Alla luce degli esempi precedenti è importante comprendere che tutto il codice presente in un progetto Instant Developer Cloud sarà in funzione nel container che ospita l'applicazione. Negli esempi precedenti, ad esempio, il codice che gestisce l'evento *onClick* è in esecuzione nel server Node.js, non nel browser dell'utente.

Questa soluzione ha diversi effetti positivi:

1) Tutto il codice applicativo, anche quello di gestione del front-end, è in grado di usare le risorse del back-end, quindi, ad esempio, può fare una query sul database.
2) Non è necessario creare uno strato di webapi per far funzionare il front-end, quindi l'applicazione è più sicura oltre a richiedere molto meno tempo di programmazione.
3) Il server opera sull'applicazione sempre in modalità *push*, quindi quando vengono rilevati eventi non derivanti dall'interfaccia utente, il server può comunque aggiornare il browser senza alcun intervento dell'utente.

È possibile usare Instant Developer Cloud anche per creare architetture più tradizionali, cioè lasciare in esecuzione nel cloud uno strato di webapi e sviluppare il front-end in modalità offline, cioè completamente in esecuzione nel browser o nel device. In questo caso il front-end ed il back-end potranno comunicare tramite webapi.

Il sistema di sincronizzazione di Instant Developer Cloud è in grado di convertire automaticamente l'architettura standard vista in precedenza in questo secondo tipo. Si consiglia quindi di utilizzare Instant Developer Cloud in modalità standard avendo comunque la possibilità di scegliere successivamente il tipo di architettura finale della propria applicazione.

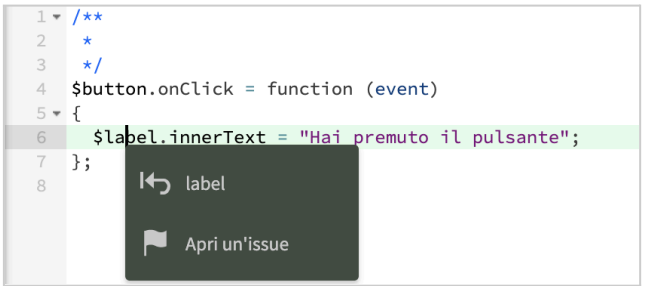
## Referenziare elementi dal codice della videata

Come abbiamo visto in precedenza, il codice inserito all'interno di una videata può riferirsi alla sessione applicativa tramite la variabile *app*. È altresì possibile riferirsi alla videata attuale usando la variabile *view*, sempre definita all'interno degli script della stessa.

Sia la videata che tutti i suoi elementi possono riferirsi agli elementi contenuti, come proprietà dell'oggetto stesso. Ad esempio, nel caso della videata nell'immagine precedente è possibile riferirsi alla label di cui cambiare il testo tramite la seguente espressione:

```
view.IonStdPage.content.list.item.label
```

Per facilitare la scrittura del codice, l'editor di codice supporta la seguente sintassi: *$<nome elemento>*, quindi nel caso precedente è possibile scrivere semplicemente *$label*. Se la videata contiene più elementi con lo stesso nome, viene referenziato il "più vicino" allo script che si sta scrivendo, intendendo quello contenuto nel contesto più piccolo che contiene anche lo script stesso. Se ci sono più elementi con la stessa distanza, verrà selezionato il primo. Per controllare a quale elemento si riferisce una determinata espressione, utilizzare il menu contestuale dell'editor di codice, come mostrato nell'immagine seguente.



# Videate come componenti

Abbiamo visto che per definire l'interfaccia utente di una videata è necessario aggiungere gli elementi visuali che la compongono. Tali elementi possono corrispondere uno a uno con gli oggetti DOM del browser, ma possono rappresentare anche oggetti più complessi come mappe, grafici, slider, liste e così via. Abbiamo visto anche che è possibile estendere gli elementi disponibili importando il relativo codice JavaScript e definendo una classe di interfaccia.

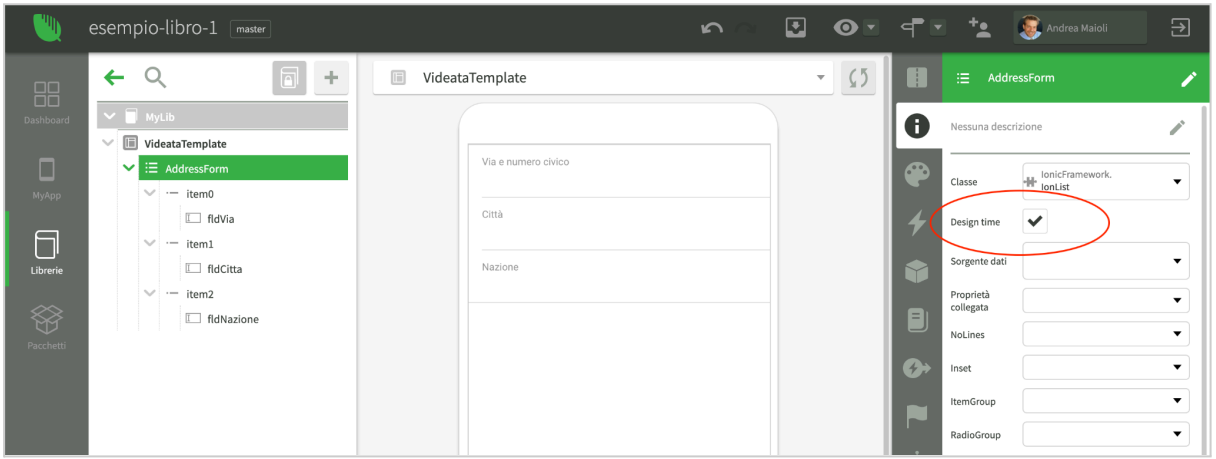
Tuttavia in diverse occasioni è interessante poter definire nuovi componenti per aggregazione di quelli esistenti. Instant Developer Cloud supporta questa modalità di creazione di componenti in due modi:

1) Tramite la definizione di template
2) Rendendo una videata disponibile come componente di design time.

## Uso dei template

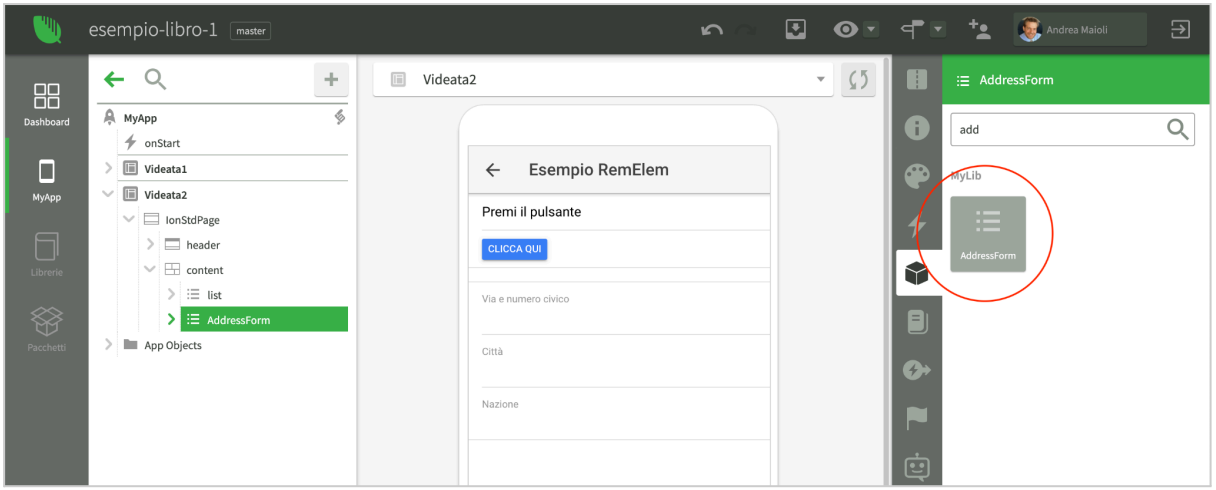
Un template è un insieme di elementi visuali che possono essere riutilizzati in contesti diversi. Viene definito inserendo una videata in una libreria personalizzata del progetto e poi attivando il flag *design time* dell'elemento che contiene il template. È necessario che la videata non abbia essa stessa tale flag attivo.

Nell'immagine seguente vediamo come esempio la lista *AddressForm* che è stata definita come template.




A questo punto è possibile utilizzare il template in ogni altra videata del progetto, come si vede nell'immagine seguente, semplicemente inserendo il template che appare nella lista degli elementi utilizzabili (pannello a destra).

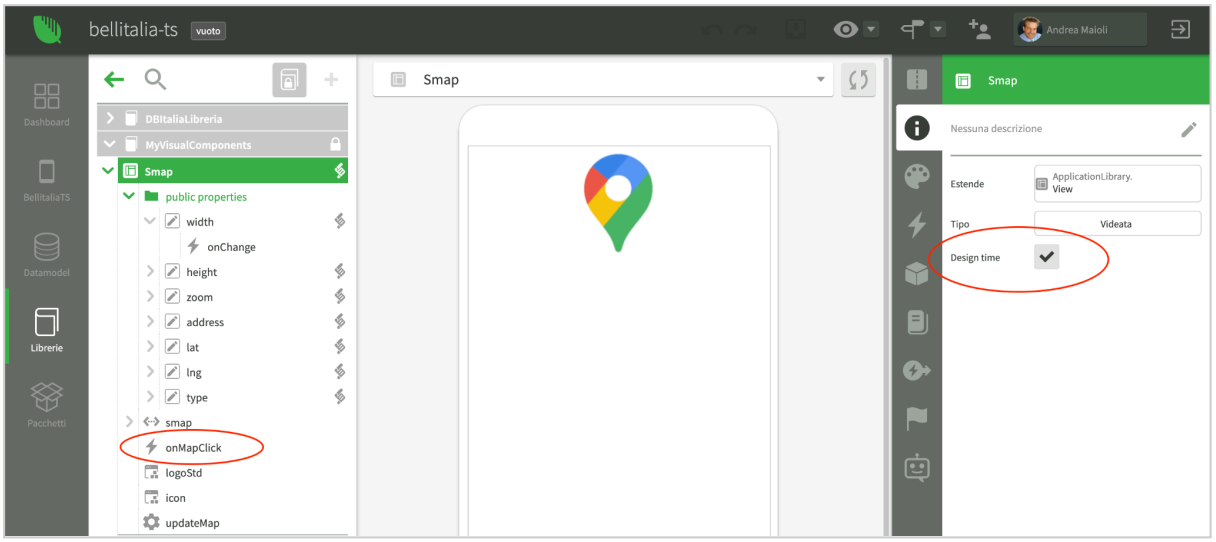
Occorre notare, infine, che le istanze di template non rimangono collegate alla sua definizione. Modificando il template, le istanze esistenti non vengono modificate.




## Usare una videata come un componente

Come abbiamo visto, i template sono semplicemente un mezzo per inserire nelle videate gli elementi visuali più velocemente. Per ottenere davvero nuovi componenti per aggregazione di quelli esistenti, è disponibile un secondo meccanismo che consiste nell'attivare il flag *design time* a livello di videata invece che di elemento visuale.

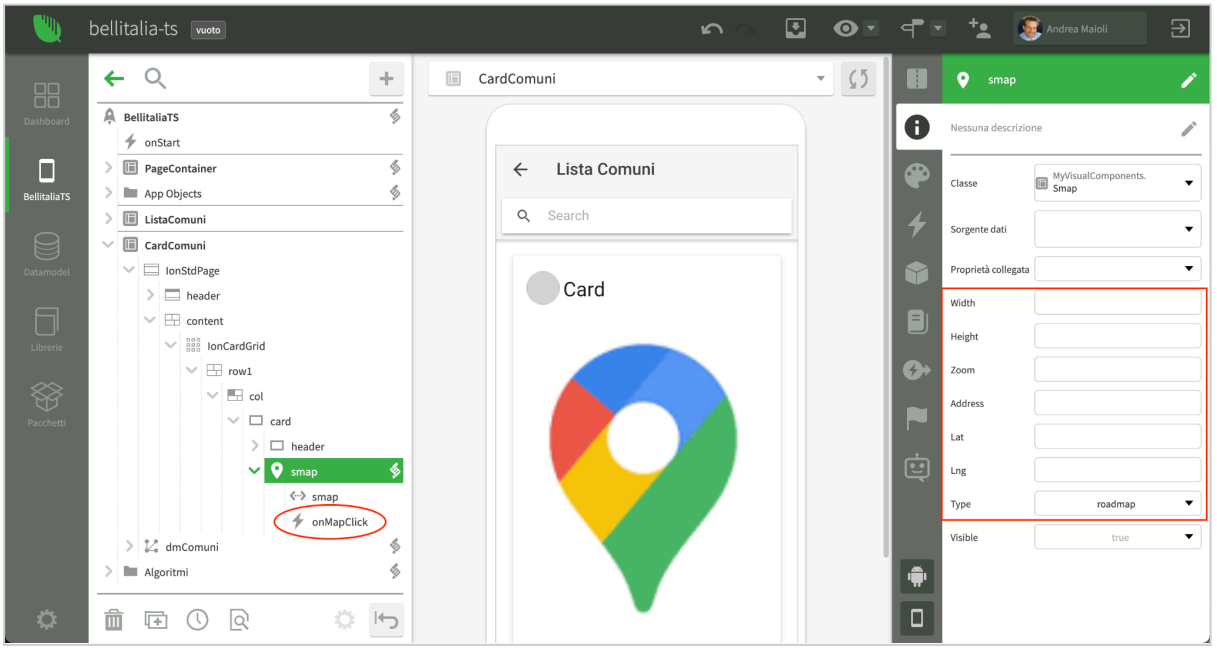
Nell'immagine seguente vediamo che per la videata Smap (Static map) è stato attivato il flag *design time*. In questo caso l'intera videata diventa un vero e proprio componente riutilizzabile che viene istanziato come se fosse un elemento visuale e diventa parte della videata che lo contiene.




Poiché Smap è un componente completo, se le proprietà definite nella videata sono state attivate come *design time* esse diventano parte dell'interfaccia del componente. Inoltre supportano l'evento *onChange* che viene notificato quando il contenitore le aggiorna.

Notiamo infine l'evento *onMapClick*, inserito come metodo della videata di tipo evento. Anche questo evento, essendo stato definito come *design time*, diventerà parte dell'interfaccia del componente.

Vediamo quindi un esempio di utilizzo del componente Smap in una videata del progetto.




In questa immagine vediamo il componente Smap contenuto all'interno di una lista di card. Sulla destra possiamo notare che le proprietà definite come *design time* sono ora impostabili dall'IDE come quelle degli elementi visuali.

Inoltre è stato definito lo script che gestisce l'evento *onMapClick*, visto che esso è definito come evento disponibile a *design time*.

Notiamo infine che, a differenza dei template, quando si usa un componente basato su una videata si crea un'istanza della classe, quindi aggiornando la classe tutte le videate si aggiornano di conseguenza.

In sintesi, siccome le librerie sono facilmente trasportabili da un progetto ad un altro tramite la definizione di pacchetti, la possibilità di definire template e componenti per aggregazione permette di creare una serie di componenti facilmente riutilizzabili in tutti i propri progetti.

# Classi e librerie

Nei paragrafi precedenti abbiamo definito due tipi di classi presenti nel progetto: la sessione (*app*) e le videate (*view*). Il primo tipo viene gestito autonomamente dal framework, il secondo viene usato per definire l'interfaccia utente dell'applicazione.

In un progetto Instant Developer Cloud è possibile definire anche altri tipi di classi, sia specializzate che generiche. Le classi specializzate vengono utilizzate per gestire in modo strutturato i dati del database o l'interfaccia verso le webapi, e saranno trattate nei capitoli relativi.

Le classi di codice generiche possono essere definite sia a livello di applicazione che all'interno di una libreria. Le librerie sono dei sottoinsiemi riutilizzabili di elementi del progetto; il contenuto di una libreria può essere referenziato da tutte le applicazioni del progetto ed è facilmente esportabile come pacchetto per essere incluso in altri progetti.

Una classe di codice generica è una classe JavaScript basata su *prototype* e può contenere proprietà di classe e metodi, sia statici che di istanza. Le proprietà, invece, sono definite solo a livello di istanza, in quanto le proprietà statiche sarebbero condivise fra tutte le sessioni del container generando effetti indesiderabili.

Per referenziare una classe definita a livello di applicazione è sufficiente usare l'oggetto *App*, ad esempio:

```
let m = new App.MyClass(app);
```

Se invece la classe è contenuta in una libreria, si deve anteporre anche il nome di quest'ultima. Ad esempio:

```
let m = new App.MyLibrary.MyClass(app);
```

## Tipi di librerie e di classi

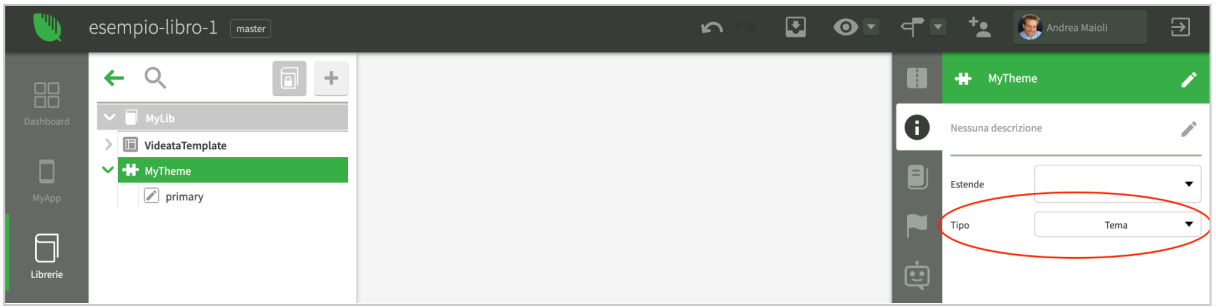
Sono disponibili tre diversi tipi di librerie, selezionabili dalla videata delle proprietà delle stesse:

1) *Applicazione*: la libreria definisce le interfacce con il framework server o client. Queste librerie non contengono codice specifico ma solo le definizioni; solitamente vengono importate con i pacchetti base del progetto.
2) *Datamodel*: questo tipo di libreria definisce l'interfaccia con un determinato tipo di database (o di sistema di memorizzazione dati). Solitamente non vengono mai modificate.
3) *Personalizzata*: questo tipo di librerie contiene elementi e codice specifico del progetto in cui sono contenute. È il tipo di default e normalmente è quello che si desidera utilizzare nei propri progetti.

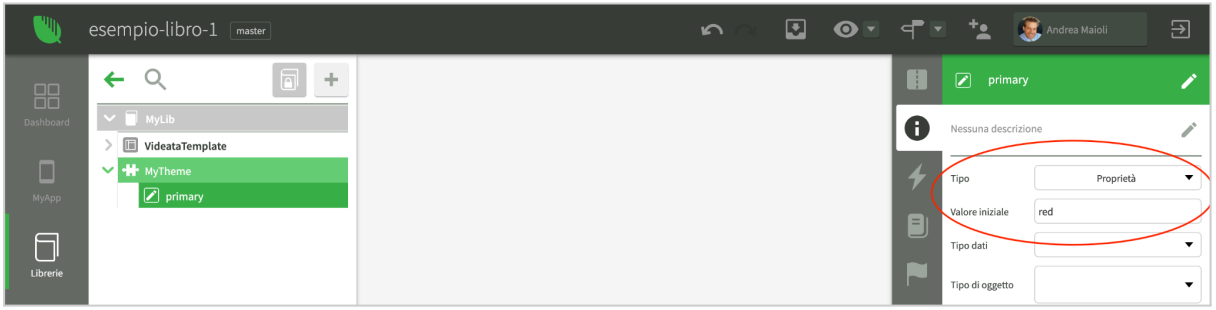
Anche le classi generiche possono essere di diversi tipi:

1) *Personalizzata*: questo tipo di classi sono le più comuni e contengono il codice applicativo specifico del progetto in cui sono contenute.

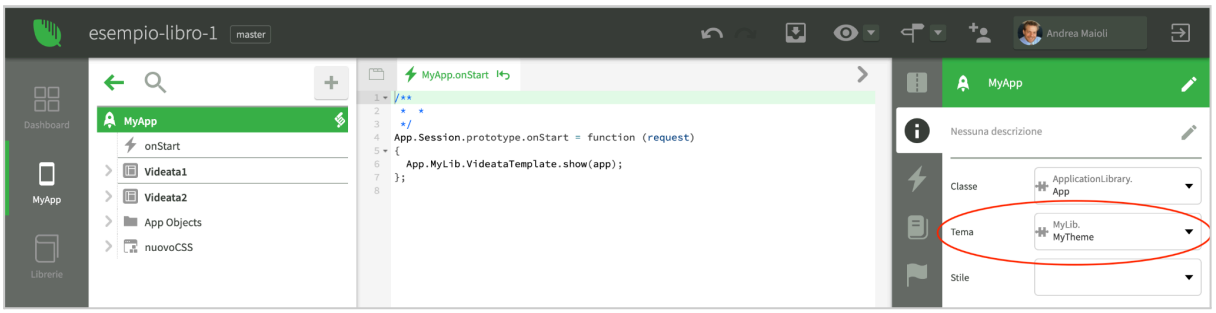
2) *Stile / Tema*: questo tipo di classi servono per personalizzare il tema grafico e lo stile dell'applicazione. In particolare le classi di tipo tema permettono di personalizzare a design time le proprietà del tema dell'applicazione. A tal fine, la proprietà deve avere lo stesso nome di quella del tema e la classe deve essere indicata come la classe "tema" nelle proprietà dell'applicazione.



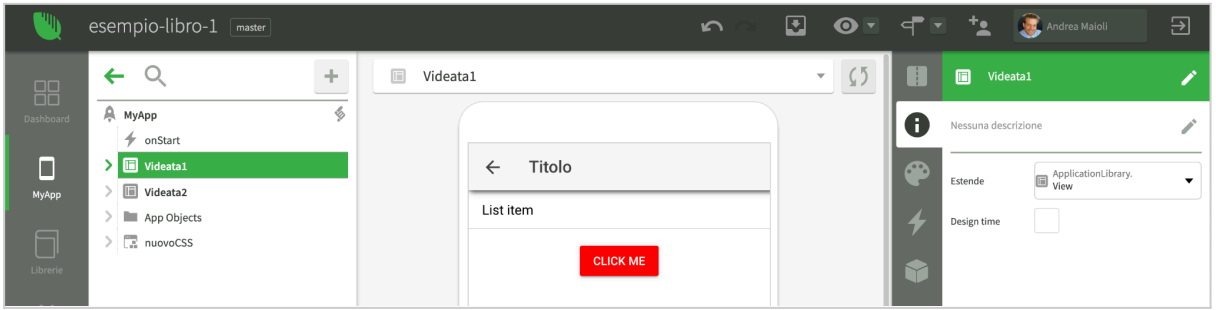

*Creazione della classe di tipo tema*




*Impostazione della proprietà* primary




*Impostazione di* MyTheme *come tema dell'applicazione*




*Ora il colore* primary *è definito come* red

3) *Elemento*: le classi di tipo elemento definiscono l'interfaccia verso un elemento visuale di uno specifico tipo. Per aggiungere al progetto componenti visuali non presenti nelle librerie di tipo è necessario definire una classe di tipo *Elemento* all'interno di una libreria di tipo *Applicazione*.

4) *Videata*: definisce la classe base di tutte le videate. Non deve essere né usata né modificata. Solitamente viene importata con i pacchetti base del progetto.
5) *Documento*: definisce la classe base di tutti i documenti. Non deve essere né usata né modificata. Solitamente viene importata con i pacchetti base del progetto.
6) *Datamodel*: definisce un'interfaccia verso un database. Deve essere contenuta in una libreria di tipo Datamodel. Solitamente non viene utilizzata, ma solo importata con i pacchetti base del progetto.

# Proprietà e passaggio di parametri

In questo paragrafo abbiamo visto che tutti gli elementi costitutivi di un progetto Instant Developer Cloud sono classi, anche la sessione e le videate.

Ognuna di queste classi può avere proprietà specifiche, inseribili tramite i comandi dell'IDE. Può essere interessante esemplificare il miglior modo di condividere il valore delle proprietà fra le istanze di queste classi.

## Proprietà di sessione (applicazione)

Le proprietà inserite a livello dell'oggetto applicazione vengono memorizzate a livello di ogni singola sessione e possono essere referenziate in tutto il codice applicativo tramite l'espressione *app.<nome proprietà>*.

Un esempio di proprietà che di solito si aggiunge alla sessione è l'oggetto che descrive l'utente collegato. Se ad esempio questa proprietà si chiama *loggedUser* e la tabella di database corrispondente ha un campo *id,* si potrà accedere a tale valore in tutta l'applicazione scrivendo *app.loggedUser.id*.

## Proprietà di videata

Le proprietà di videata sono visibili all'interno del codice della videata stessa tramite l'espressione *view.<nome proprietà>*. Solitamente vengono passate alla videata quando questa si apre tramite le opzioni del metodo di apertura. La videata riceve i parametri nell'evento *onLoad*.

## Proprietà di documento

Un documento è una classe speciale che rappresenta un'entità del framework ORM di Instant Developer Cloud. Le proprietà di un documento possono essere impostate nel costruttore passandole come oggetto. Ad esempio:

```
var p = new App.BE.Product(app, {ProductName:"Chai"});
```

## Proprietà di classe

Le proprietà di una classe generica sono visibili all'interno del codice della classe stessa. Le proprietà possono essere impostate dopo aver costruito l'oggetto chiamando un metodo specifico.

# Risorse e CSS

Un ulteriore elemento dei progetti Instant Developer Cloud sono le risorse: file di diverso contenuto che vengono utilizzati dal codice dell'applicazione come dati, elementi grafici, personalizzazioni, eccetera.

La creazione di una risorsa avviene generalmente trascinando il file dal proprio desktop nell'albero degli oggetti del progetto. In quel momento il file viene caricato nel cloud e diventa parte del progetto.

Se la risorsa viene caricata all'interno di una specifica applicazione, verrà inclusa quando l'applicazione viene installata. Se invece viene inserita in una libreria, essa verrà resa disponibile in ogni applicazione contenuta nel progetto.

Le risorse sono di tipo diverso e, in base al tipo, hanno comportamenti specifici. Vediamo ora una rassegna dei tipi di risorsa disponibili.

## Risorse di tipo CSS

Le risorse di tipo CSS contengono definizioni di stili aggiuntive a quelle standard del progetto. Per creare una risorsa di tipo CSS è possibile trascinare un file CSS esistente nel progetto, oppure crearne uno vuoto tramite il menu contestuale dell'oggetto applicazione.

Se il file CSS viene creato all'interno dell'IDE, esso viene considerato come stringa CSS e non viene salvato come file vero e proprio. In ogni caso viene incluso nella pagina web dell'applicazione.

Si consiglia di inserire risorse CSS solo a livello di applicazione o di libreria. Si tenga conto che, anche creandole a livello di videata o di classe, esse verranno applicate fin dall'inizio della sessione.

Alcuni file CSS vengono inclusi nel progetto al fine di aggiungere icone invece che definizioni di stile. In tal caso si consiglia di indicare come tipo di contenuto il valore *Icona*, così sarà possibile selezionare le classi corrispondenti tramite gli strumenti dell'IDE dedicati alla selezione visuale delle icone.

## Risorse di tipo immagine

Le risorse di tipo immagine rappresentano elementi grafici utilizzabili nel progetto. Dopo aver caricato una risorsa immagine, essa può essere referenziata nei file CSS del progetto, negli stili o nel codice utilizzando la notazione *$<nome>*, come mostrato nell'esempio seguente.

```
.mycls {
  background-image: url('$logo');
}
```

## Risorse di tipo font

Vengono utilizzate per aggiungere la definizione di un nuovo font all'applicazione. Come nome della risorsa occorre indicare il nome del font nella stessa forma della dichiarazione *font-family*. Come indirizzo del font, indicare l'URL del file CSS che ne specifica l'import.

Per importare un font da Google Fonts, si consiglia di ottenere la direttiva di importazione, come mostrato nell'esempio seguente per il font *Otomanopee One*:

```
Otomanopee One
<style>
@import
url('https://fonts.googleapis.com/css2?family=Otomanopee+One&display=swap')
;
</style>
```

La risorsa font da creare dovrà quindi avere come nome *Otomanopee One* e come URL *https://fonts.googleapis.com/css2?family=Otomanopee+One*. A questo punto potrà essere utilizzata sia nei file di tipo CSS che nella barra di stile dell'IDE.

Per fare in modo che il font sia disponibile anche per applicazioni offline, è necessario utilizzare il comando *Crea copia locale* nel menu contestuale della risorsa. Tale comando converte in risorse locali tutti i riferimenti remoti al font, scaricando anche i file binari di definizione. Prima di effettuare questa operazione, è necessario verificare di avere i permessi di licenza necessari.

## Risorse di tipo definizione SVG

Le risorse di tipo definizione SVG contengono un file di testo che specifica le icone in formato SVG disponibili per l'applicazione. È importante specificare il valore *Icona* come tipo di contenuto della risorsa in modo da poter selezionare le varie icone tramite gli strumenti dell'IDE.

Un esempio di file di definizione icone è il seguente:

```
<svg display="none" version="1.1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <symbol id="icon-logo" viewBox="0 0 40 40">
      <path d="M32 11.7v15..."/>
    </symbol>
    <symbol id="icon-main-dashboard" viewBox="0 0 40 40">
      <path d="M33.923 35h-10.77C21.724 ..." fill-rule="evenodd"/>
    </symbol>
 </defs>
</svg>
```

Esistono diversi strumenti online per la composizione dei file di definizione icone, ad esempio: icomoon.

## Risorse di tipo ACS

Una risorsa di tipo ACS (Access Control Sheet) contiene le specifiche degli elementi dell'interfaccia utente in funzione dei ruoli, del nome dell'utente o della lingua della sessione. Tramite queste risorse è possibile personalizzare in modo dichiarativo le proprietà degli elementi dell'interfaccia utente, attivando o disattivando le varie funzioni in relazione all'utente collegato. Per maggiori informazioni su questa funzionalità è possibile vedere il progetto acs-design-patterns.

## Risorse di tipo HTML

Una risorsa di tipo HTML rappresenta un nuovo file HTML disponibile nel progetto. Tale risorsa può essere referenziata con la notazione *$<nome>* nel progetto.

## Risorse di tipo testo

Una risorsa di tipo testo rappresenta un file di testo disponibile nel progetto. Tale risorsa può essere referenziata con la notazione *$<nome>* nel progetto. Il seguente codice, ad esempio, legge e converte in oggetto un file di testo in formato JSON caricato come risorsa.

```
// Lettura file JSON
var f = app.getResourceFile($italy_regions);
yield f.open();
var s = yield f.readAll();
yield f.close();
//
// converto in oggetto
var list = JSON.parse(s);
console.log(list);
```

## Risorse di tipo file

Sono simili alle risorse di tipo testo, ma contengono un file di tipo generico..

## Risorse di tipo file server

Le risorse di tipo file server vengono installate nella root directory dell'applicazione, sovrascrivendo eventualmente quelle esistenti. In questo modo è possibile caricare, ad esempio, un file *index.html* personalizzato.

Queste risorse devono essere utilizzate con cautela in quanto possono invalidare il funzionamento dell'applicazione. Inoltre, se si aggiorna il server IDE o il server di produzione ad una differente versione di Instant Developer Cloud, occorre nuovamente verificare che le proprie risorse di tipo file server siano allineate con i funzionamenti della versione appena installata.

## Risorse di tipo script client

Le risorse di tipo script client vengono usate per aggiungere al progetto elementi visuali personalizzati. È possibile fare riferimento al progetto extensibility-design-patterns per un esempio di utilizzo.

### Risorse di tipo script server

Le risorse di tipo script server vengono usate per aggiungere codice di base al framework del progetto. Il codice contenuto potrà essere utilizzato nel codice applicativo. È possibile fare riferimento al progetto extensibility-design-patterns per un esempio di utilizzo.

Si ricorda che è possibile aggiungere al proprio server IDE o di produzione anche pacchetti Node.js personalizzati tramite la console di controllo. Dopo aver aggiunto tali pacchetti il codice corrispondente risulta subito disponibile nel codice applicativo, anche se si dovrà essere certi che tale codice verrà eseguito solo nel container per applicazioni web e non come sessione offline.

### Risorse di tipo plugin

Le risorse di tipo plugin contengono gli script di interfaccia per un plugin nativo Cordova personalizzato. È possibile fare riferimento al progetto extensibility-design-patterns per un esempio di utilizzo.

### Risorse di tipo script IDE

Le risorse di tipo script IDE vengono usate per aggiungere un plugin personalizzato all'IDE di Instant Developer Cloud. Questi plugin possono modificare il funzionamento dell'IDE a 360° quindi si consiglia prudenza nell'utilizzo. È possibile fare riferimento al progetto extensibility-design-patterns per un esempio di utilizzo.

### Risorse di tipo lingua

Una risorsa di tipo lingua contiene i dati linguistici creati per l'applicazione tramite il sistema di gestione delle traduzioni di Instant Developer Cloud. Si consiglia di non modificare direttamente il contenuto di questa risorsa in quanto si potrebbe invalidare il sistema di gestione automatica delle traduzioni dell'interfaccia utente.

# I pacchetti

Un elemento indispensabile nello sviluppo di applicazioni è la possibilità di condividere parti di un progetto per poterle utilizzare in altri progetti. A questo scopo è possibile utilizzare i pacchetti di Instant Developer Cloud (detti anche componenti) che è possibile creare all'interno di un progetto ed esportare e utilizzare in tutti gli altri progetti della propria organizzazione.

Per creare un pacchetto è necessario cliccare sull'icona *Pacchetti* nella toolbar di sinistra all'interno dell'IDE. In questo modo si entra nella relativa sezione del progetto. Occorre quindi cliccare sul pulsante più (+) per creare un nuovo pacchetto.
Per aggiungere elementi al pacchetto è necessario tornare nell'albero del progetto e selezionare gli elementi che si desidera inserire al suo interno. Occorre quindi cliccare sul pulsante a forma di ingranaggio in fondo all'albero, che permette di aggiungere gli oggetti selezionati al pacchetto.

Possono essere aggiunti ad un pacchetto i seguenti elementi: librerie, classi, videate e database.

Il pacchetto deve avere un nome, una descrizione e altre proprietà che possono essere modificate utilizzando il pulsante a forma di matita presente sul pacchetto stesso.
Di seguito vengono indicate in dettaglio tutte le proprietà e il loro significato.

Il parametro *Nome* indica il nome con il quale verrà visualizzato il pacchetto nella sezione pacchetti. Il nome non può contenere spazi o trattini. È ammesso l'utilizzo di maiuscole e minuscole.

Il parametro *Descrizione* indica la descrizione con la quale verrà visualizzato il pacchetto nella sezione pacchetti. Questo campo può essere utilizzato per indicare lo scopo e le modalità di utilizzo del pacchetto.

Il parametro *Versione* indica il numero di versione del pacchetto.

Il parametro *Versione IDE* indica la versione che deve avere l'IDE di Instant Developer Cloud per poter importare il pacchetto. Se questo parametro non è indicato, la versione non viene controllata.
Questo parametro si può impostare in vari modi:

- "23.5" => indica che il pacchetto è utilizzabile solo con la versione 23.5
- "21.0:23.0" => indica che il pacchetto è utilizzabile dalle versioni comprese tra 21.0 (inclusa) e 23.0 (inclusa).
- "23.0:" => indica che il pacchetto è utilizzabile da tutte le versioni a partire dalla 23.0 (inclusa).
- ":23.0" => indica che il pacchetto è utilizzabile da tutte le versioni fino alla 23.0 (inclusa).

Il parametro *Prerequisiti* indica i nomi dei pacchetti che devono essere importati prima del pacchetto in questione come requisito di funzionamento. Se sono presenti più nomi occorre separarli mediante il carattere ";" (punto e virgola).
In fase di importazione, se il pacchetto richiesto non è già stato importato, viene visualizzato un messaggio di errore e il pacchetto non viene importato.

Il parametro *Lingua* indica la lingua in cui verrà importato il pacchetto.
In fase di esportazione si possono indicare le descrizioni in più lingue.
In fase di importazione il sistema cerca nel file delle traduzioni la lingua in cui il pacchetto viene importato e aggiusta le descrizioni.

Il parametro *Aggiornamento automatico* non è gestito per il momento e quindi non occorre compilarlo (è stato aggiunto per una implementazione futura).

Il parametro *Esporta sorgenti*, se impostato a *true,* indica che vengono esportati anche i sorgenti, ovvero i metodi del pacchetto conterranno tutti i token singoli che puntano ai rispettivi oggetti.
Se i sorgenti non vengono esportati, tutti i token vengono eliminati e ogni metodo ricorda solo il codice intero come un'unica stringa. Per esempio, la funzionalità *"trova dove usato"*,

non troverà riferimenti all'interno del componente. I token sono le chiavi che identificano ogni elemento dei sorgenti all'interno dei progetti di Instant Developer Cloud.
Attenzione: se non vengono esportati i sorgenti, in fase di esportazione viene memorizzato nei metodi l'intero codice che sarà generato quando si compila (che non è lo stesso visualizzabile nell'editor di codice).

Il parametro *Personalizzabile*, se impostato a *true,* indica che è permesso lo sblocco del pacchetto e, di conseguenza, che il contenuto dello stesso può essere liberamente modificabile.
Questo implica che, se il pacchetto viene importato nuovamente, si perderanno le modifiche effettuate nella versione modificata. Lo sblocco del pacchetto è possibile anche se non sono stati esportati i sorgenti completi.
Per sbloccare un pacchetto e poterlo modificare una volta importato, è necessario cliccare sul pulsante a forma di lucchetto presente nel pacchetto.

A questo punto il pacchetto è pronto e può essere esportato dalla sezione dei pacchetti dell'IDE per renderlo disponibile per l'importazione nei propri progetti.

Una volta esportato un pacchetto, questo è visibile al solo utente che lo ha creato nella sezione *Componenti* della Console di Instant Developer Cloud.
Un pacchetto può essere condiviso con la propria organizzazione, con uno o più gruppi di utenti o solo con alcuni utenti della stessa.
Un pacchetto può anche essere reso pubblico e visibile a tutti gli utenti di Instant Developer Cloud.
Gli utenti coinvolti nella condivisione avranno la possibilità di importare il pacchetto.

# Programmazione asincrona

Concludiamo questo capitolo sulla struttura di un'applicazione Instant Developer Cloud descrivendo il modello di multitasking previsto per i vari tipi di container.

Come previsto dall'architettura delle virtual machine JavaScript, sia i container basati su Node.js, sia quelli basati su browser o webview (Cordova) utilizzano il meccanismo delle closure e delle callback per la gestione delle operazioni asincrone e per il multitasking cooperativo.

Questo significa che ogni operazione asincrona che coinvolge, ad esempio, query su database, lettura di dati dal file system o chiamate remote deve essere aspettata e gestita in un ramo di codice separato tramite una callback JavaScript.

Per semplificare la scrittura del codice, in particolare quello in cui si hanno diverse operazioni asincrone in cascata, Instant Developer Cloud include fino dal 2015 la sequenzializzazione automatica delle operazioni asincrone tramite il costrutto *yield*.

Ad esempio, se volessimo stampare nella console i numeri da 0 a 9 ad un intervallo di un secondo l'uno dall'altro, potremmo scrivere le seguenti righe di codice:

```
for (let i = 0; i < 10; i++) {
  console.log(i);
  yield app.sleep(1000);
}
```

Il metodo *app.sleep* del framework sospende il thread di codice attivo per un determinato intervallo di tempo e poi lo riprende. L'inserimento automatico della parola chiave *yield* consente di aspettare il termine dell'operazione asincrona prima di riprendere l'esecuzione dello script attivo.

Il risultato di questa sequenzializzazione è logicamente identico all'odierno costrutto *await*, con le seguenti differenze notevoli:

1) L'IDE rileva automaticamente se una riga di codice chiama un metodo marcato come asincrono. In tal caso prima della chiamata viene automaticamente inserito il costrutto *yield* e il metodo corrente viene marcato come asincrono.
2) Quando un metodo viene marcato come sincrono o asincrono (in funzione del fatto che al suo interno vengano sequenzializzate chiamate a metodi o meno), tutti i metodi che chiamano quello in fase di modifica vengono automaticamente modificati aggiungendo o togliendo il costrutto *yield* ove necessario.
3) È possibile scegliere di non sequenzializzare una chiamata eliminando la parola chiave *yield* dopo che l'IDE l'ha inserita. In tal caso verrà aggiunta una callback per la gestione dei risultati della chiamata.
4) Il codice dei metodi asincroni e le relative chiamate vengono generati in modo tale da gestire le chiamate asincrone. Non è necessario alcun intervento di modifica manuale.
5) La gestione delle eccezioni relative ad una chiamata asincrona è identica a quella di un metodo sincrono. Non è necessaria alcuna modifica al codice applicativo.

Si deve tenere presente che durante l'attesa del completamento di un metodo asincrono il sistema non è bloccato, quindi è possibile che avvengano azioni che possono avviare ulteriori processi di codice. Se, ad esempio, lo script di gestione del clic di un pulsante richiede 10 secondi, è possibile che l'utente clicchi più volte il pulsante prima del termine del processo, causando l'avvio di più istanze dello stesso processo di gestione. In questi casi si consiglia di disabilitare il comando fino al completamento del processo, oppure di applicare un elemento di attesa all'interfaccia dell'applicazione bloccando tutti gli input dell'utente.

Notiamo infine che, nel caso di container per applicazioni web, nello stesso processo worker Node.js saranno gestite più sessioni, solitamente una quantità da 10 a 100. Se una di queste sessioni esegue un metodo sincrono molto impegnativo, tutte le sessioni dello stesso processo worker verranno rallentate. Non è mai consigliabile che uno script sincrono richieda più di 100 ms per essere eseguito. In questi casi è possibile isolare gli script sincroni lenti in un processo separato ad esempio utilizzando una sessione di tipo *server*, oppure suddividere il lavoro in più step sincroni separati da chiamate asincrone che permettono alle altre sessioni di continuare il proprio lavoro.