# Instant Developer Cloud

## Indice generale

# Instant Developer Cloud

La piattaforma cloud nativa per sviluppare, distribuire e gestire i sistemi di trasformazione digitale

# Introduzione

Siamo nell'era della trasformazione digitale, in cui ogni processo viene rivisto e ottimizzato tramite il supporto di tecnologie informatiche. E per *ogni processo*, si intendono tutti i processi aziendali e industriali, ma anche quelli correlati alla persona, nella vita quotidiana.

Normalmente i sistemi di trasformazione digitale hanno alcune caratteristiche comuni che li contraddistinguono:

1) Sono basati su infrastruttura cloud: la raccolta globale dei dati relativi al processo è infatti il presupposto per la sua ottimizzazione, anche attraverso l'uso di intelligenza artificiale.
2) Sono di tipo "omnichannel", cioè la fruizione del servizio deve essere possibile in multicanalità: ad oggi si devono considerare almeno i browser e i principali dispositivi smartphone e tablet, ma nel prossimo futuro anche chatbot intelligenti e ar/vr.
3) Sono integrati con i principali canali di accesso del mondo digitale e quindi con:
    a) I social network
    b) I sistemi di notifica e le email
    c) I motori di ricerca

## Le rivoluzione digitale

Vediamo ora quali sono le principali problematiche da affrontare per lo sviluppo di un sistema di trasformazione digitale:

*Infrastruttura cloud e scalabilità*: un sistema di trasformazione digitale è quasi sempre basato su una infrastruttura cloud pubblica. È possibile configurare e gestire manualmente questi sistemi utilizzandoli a livello IAAS (infrastructure as a service), ma è consigliabile l'adozione di una piattaforma PAAS (platform as a service) che gestisce il cloud in modo automatico e in cui sono presenti procedure di disaster recovery, di gestione della scalabilità e di controllo dell'integrità del sistema. Per ottimizzare i costi di runtime del servizio è importante avere a disposizione un servizio di test di carico in grado di simulare i diversi livelli operativi che dovranno essere garantiti.

*UX Design e multicanalità*: i sistemi di trasformazione digitale sono spesso indirizzati ad un pubblico eterogeneo e distribuito. Occorre quindi una accurata progettazione della *user experience* in modo da rendere più semplice possibile la fruizione del servizio da parte degli utenti finali. Occorre infine prevedere lo sviluppo di un set di applicazioni che vanno dall'interfaccia web alle applicazioni mobile, ma anche quelle per i sistemi di back office e di controllo del sistema.

*Integrazioni con altri sistemi informatici*: un sistema di trasformazione digitale non vive mai isolato, anzi spesso nasce proprio per permettere la condivisione di dati che altrimenti rimarrebbero inutilizzati perché chiusi. In questo senso dovranno essere predisposte le opportune API sia come generazione che come consumo, ma occorrerà anche affrontare l'integrazione del mondo *on-premise* che di solito non ha contatti con il mondo del cloud.

*Distribuzione ed aggiornamenti*: una puntuale distribuzione dell'applicazione nell'infrastruttura cloud e negli app store è un elemento chiave del successo dell'operazione. È fortemente consigliabile che l'infrastruttura PAAS utilizzata abbia un sistema automatico di aggiornamento che includa anche gli schemi di database, altrimenti si possono perdere ore di downtime ad ogni release del prodotto. Nel contempo si deve predisporre un sistema automatico di aggiornamento delle applicazioni mobile in tempo reale, superando il tradizionale ciclo di distribuzione tramite app store.

*Controllo del corretto funzionamento del sistema*: sia durante le fasi di beta test che di effettivo utilizzo del sistema, è necessario dotarsi di un sistema di controllo che consenta di verificarne il corretto funzionamento. Per quanto riguarda i server nel cloud si consiglia di dotarsi di un sistema di analisi dei log e uno strumento di debug realtime; per quanto riguarda le applicazioni di un sistema di analytics per raccogliere sia i dati di utilizzo che le eventuali anomalie di funzionamento. Infine può essere molto utile un sistema di raccolta feedback in-app, che consente agli utenti di farci sapere cosa pensano della nostra app senza dover per forza utilizzare le recensioni negli app store.

*Sicurezza informatica*: questo argomento è oggi sempre più in evidenza, dato che la creazione di un sistema sicuro è un argomento complesso e costoso. Da questo punto di vista è consigliabile l'utilizzo di un sistema di sviluppo che massimizzi l'adozione di tecniche di programmazione sicura *by design*.

## Instant Developer Cloud

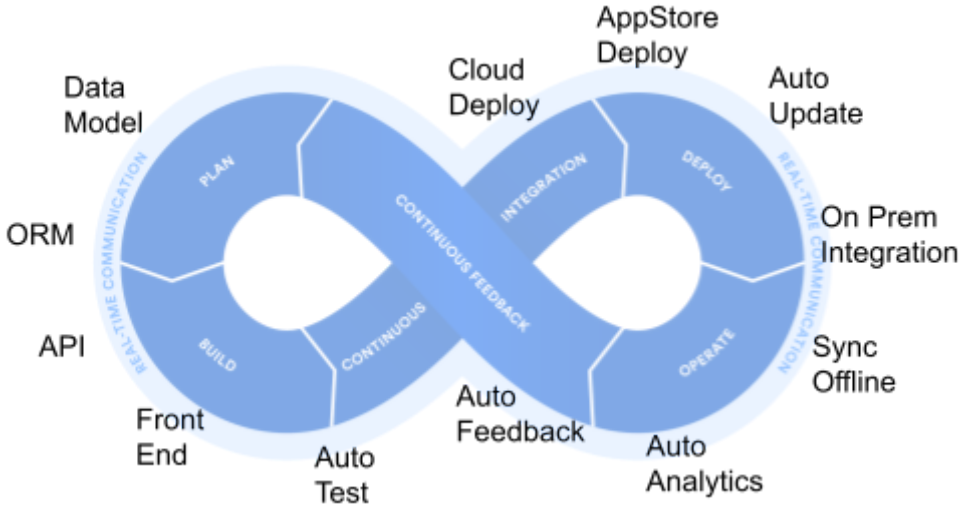
Instant Developer Cloud è la piattaforma cloud nativa nata proprio per rispondere alle esigenze delle software company e dei professionisti che vogliono creare sistemi di trasformazione digitale. Contiene tutti gli strumenti per sviluppare, distribuire e gestire tali sistemi in maniera integrata ed ottimizzata e permette di risolvere in modo efficiente tutte le problematiche indicate in questo capitolo.

Instant Developer Cloud abilita le aziende che sviluppano software ad entrare nel mondo della trasformazione digitale già da oggi, richiedendo un insieme minimo di competenze e mettendo a disposizione in modalità PAAS (platform as a service) tutti gli strumenti e i componenti necessari per il successo della propria iniziativa di trasformazione digitale.

Nei paragrafi successivi verranno analizzate la composizione della piattaforma e la struttura delle applicazioni, e al termine di questo articolo vengono riportati i prerequisiti e i supporti formativi con cui affrontare l'apprendimento dei vari componenti di Instant Developer Cloud.

# Composizione della piattaforma

La piattaforma Instant Developer Cloud contiene tutti gli strumenti per sviluppare, distribuire e gestire applicazioni omnichannel nel cloud. Il ciclo di vita di questi sistemi può essere rappresentato con il seguente schema, che si riferisce alla metodologia devops:




Il ciclo di sviluppo segue questi passaggi:

1) Definizione dei modelli dati e creazione automatica dei database relazionali nel cloud.
2) Definizione delle classi di accesso ai dati, create automaticamente all'interno del framework ORM (Object Relational Mapping) incluso. Gli algoritmi transazionali vengono implementati in linguaggio JavaScript tramite l'IDE cloud della piattaforma.
3) Definizione o importazione delle API, per l'integrazione con sistemi esterni. La definizione avviene automaticamente utilizzando il formato OData. L'importazione può essere automatica, se l'API aderisce al formato OData oppure è stata creata con Instant Developer Foundation, altrimenti avverrà in modo manuale.
4) Sviluppo del front-end a partire dal mockup o dai wireframe: avviene tramite l'IDE cloud della piattaforma; il collegamento con il back-end è automatico.
5) Realizzazione di test di non regressione tramite un sistema automatico per la registrazione e l'esecuzione di test sull'applicazione.

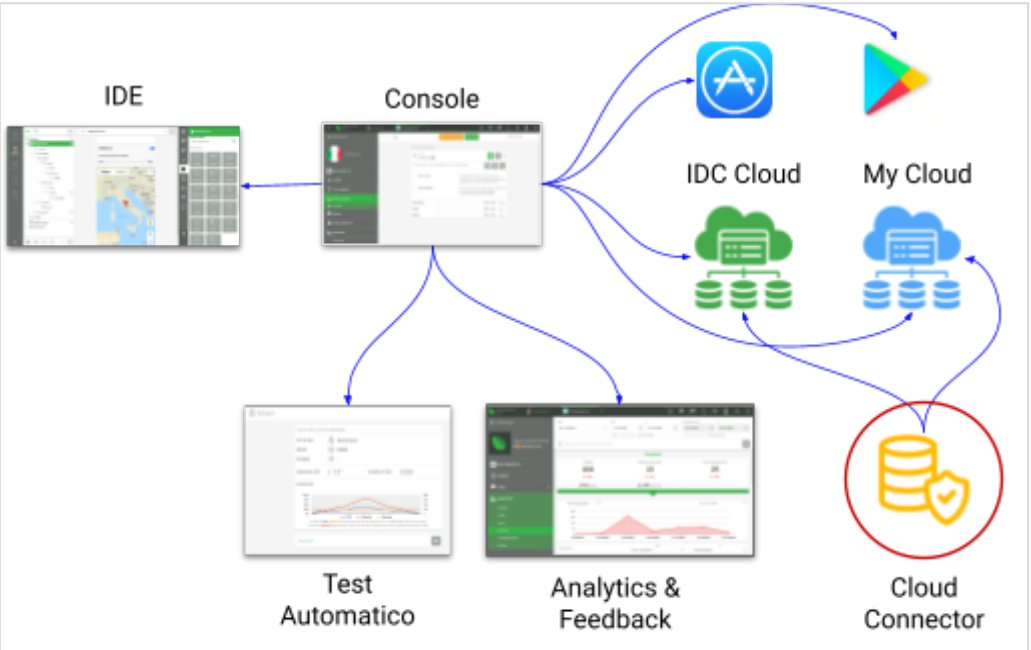
Le operazioni di distribuzione e integrazione vengono effettuate tramite la Console di Instant Developer Cloud, un vero e proprio centro di controllo delle proprie operazioni all'interno della piattaforma. Le operazioni possibili sono le seguenti:

1) La console consente l'installazione automatica sui server di produzione delle applicazioni web e dei backend. I server di produzione possono essere parte della piattaforma (Server App IDC) ma è comunque possibile utilizzare propri server, sia nel cloud che *on premise*, e gestire l'installazione in modalità manuale.
2) La console consente l'invio automatico delle applicazioni mobile ad App Store e Google Play. È possibile effettuare la compilazione e l'invio anche in modalità manuale oppure distribuire l'applicazione come PWA.
3) Per le applicazioni mobile pubblicate tramite la console, è possibile l'aggiornamento automatico senza dover passare nuovamente dagli store. L'aggiornamento è istantaneo e automatico per tutti i dispositivi in cui l'applicazione è installata.

4) Tramite un componente denominato *Cloud Connector* è possibile integrare database, file system e servizi *on premise* con il cloud. È quindi possibile usare i dati aziendali nel cloud senza dover gestire problemi di sicurezza o sviluppare software specifico. Questo componente è disponibile sui Server App IDC e sui Server My Cloud.
5) La piattaforma mette a disposizione un framework automatico per l'integrazione *client-cloud*, che permette alle applicazioni installate nei dispositivi di accedere ai dati del cloud in maniera automatica. È presente anche un sistema di sincronizzazione dei dati locali per consentire l'utilizzo anche in modalità completamente offline. Questo framework è disponibile sui Server App IDC e sui Server My Cloud.

Per quanto riguarda la gestione del software, sono presenti i seguenti strumenti:

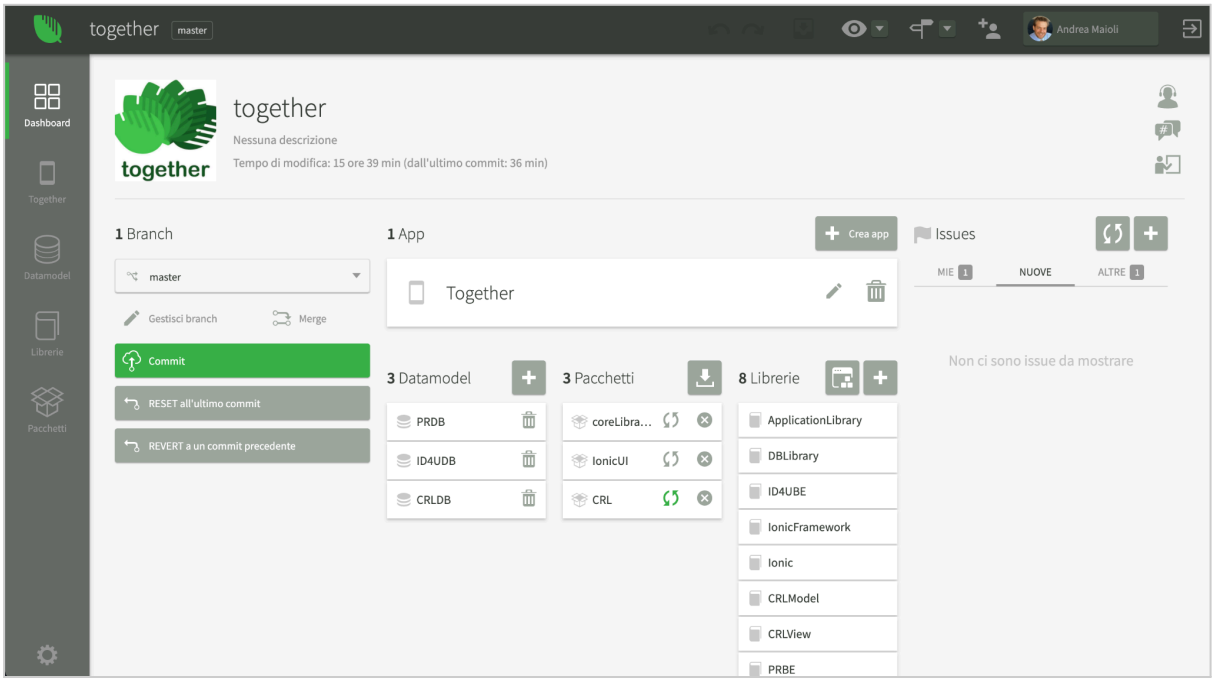
1) Per l'ottimizzazione dell'applicazione nel cloud è presente un sistema di test di carico automatico, in grado di identificare le operazioni che non scalano adeguatamente. Richiede l'utilizzo di un Server App IDC sul quale effettuare le operazioni di test.
2) Per il controllo del corretto funzionamento dei server collegati alla console, è presente un sistema di analisi dei log che li separa sessione per sessione e identifica i problemi. In questi casi è possibile anche effettuare il debug remoto di una sessione in esecuzione produzione senza dover arrestare il funzionamento del server.
3) Per il controllo del corretto funzionamento delle applicazioni web e mobile è presente un sistema di analytics completamente integrato ed automatico, in grado di tracciare il funzionamento anche se l'applicazione è offline e di rilevare come si comporta l'utente e anche le eventuali anomalie di funzionamento. Richiede l'utilizzo di un Server App IDC sul quale effettuare la raccolta dati di funzionamento.
4) È infine presente un sistema di raccolta e gestione dei feedback degli utenti che permette di inviare segnalazioni o idee direttamente dall'applicazione in modo contestuale. La console permette la gestione completa di questi feedback integrandoli nel flusso di lavoro del reparto tecnico. Richiede l'utilizzo di un Server App IDC sul quale effettuare la raccolta dei dati di feedback.




*Rappresentazione schematica della piattaforma Instant Developer Cloud*

# Struttura dei progetti

In questo paragrafo viene illustrata la struttura delle applicazioni create con Instant Developer Cloud. Nell'immagine seguente viene mostrata la dashboard di un progetto gestito nella piattaforma.




*La dashboard di un progetto nell'IDE di Instant Developer Cloud*

Come possiamo vedere nell'immagine, un progetto Instant Developer Cloud contiene le seguenti definizioni:

1) Le *App*, cioè le varie applicazioni che compongono il sistema. Ogni applicazione potrà essere compilata come applicazione web da installare nel cloud, o come applicazione mobile da inviare ad app store, o infine come PWA.
2) I *Datamodel*, cioè gli schemi dati relazionali che il sistema deve utilizzare. Possono essere collegati sia database nel cloud che on premise. Il codice SQL delle query inserito nell'IDE viene generato automaticamente per i vari tipi di database collegati (Postgres, Oracle, SQLServer, MySQL e Sqlite).
3) Le *Librerie*, cioè insiemi di classi di codice che possono essere riutilizzate fra le varie applicazioni o perfino condivise tra diversi progetti. Solitamente le librerie contengono le definizioni delle classi di accesso ai dati, classi di codice di utilità o videate standard pensate per essere riutilizzate in contesti diversi.
4) I *Pacchetti,* insiemi di componenti importati da altri progetti o definiti in quello attuale per essere esportati.

L'insieme di queste parti rappresenta un intero progetto Instant Developer Cloud, che può essere gestito da una sola persona o da un team di lavoro tramite *Teamworks*, un sistema integrato di gestione del lavoro di gruppo con funzionalità simili a *Git* (per maggiori informazioni, si legga Instant Developer Team Works). Il flusso del lavoro può essere controllato tramite un sistema di gestione di *Issue* anch'esso simile a quello presente nel sistema *GitHub*.

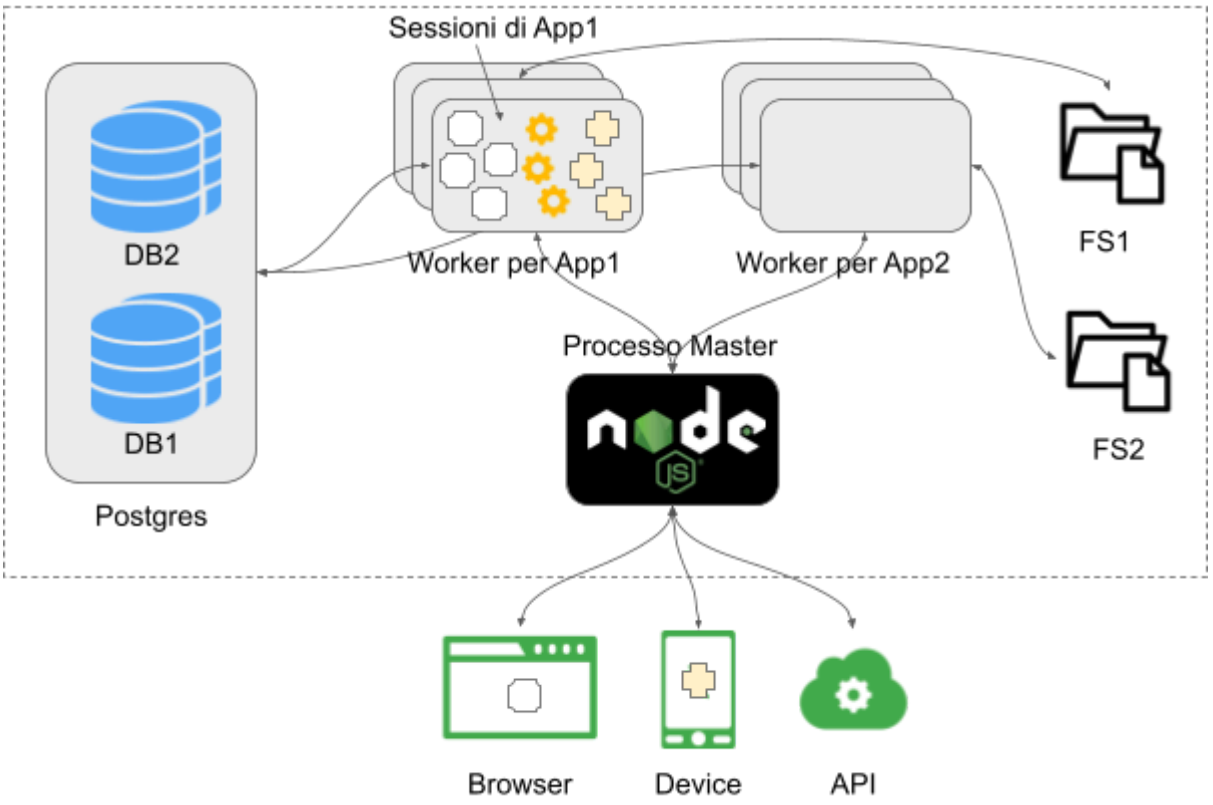
# Struttura delle applicazioni

Come abbiamo visto in precedenza, ogni applicazione presente in un progetto può essere compilata come applicazione web da installare in un server, come applicazione mobile da inviare agli store o infine come PWA.

Per consentire allo stesso codice di funzionare in maniera equivalente in ognuno di questi contesti, Instant Developer Cloud contiene diversi tipi di application container, ognuno dei quali rappresenta un device virtuale equivalente.

## Container per applicazioni web

Il container per applicazioni web è basato su un'architettura node.js multiprocesso.




Il processo master si occupa di gestire le connessioni e le sessioni. I processi worker contengono le sessioni di lavoro. Sono previsti i seguenti tipi di sessioni:

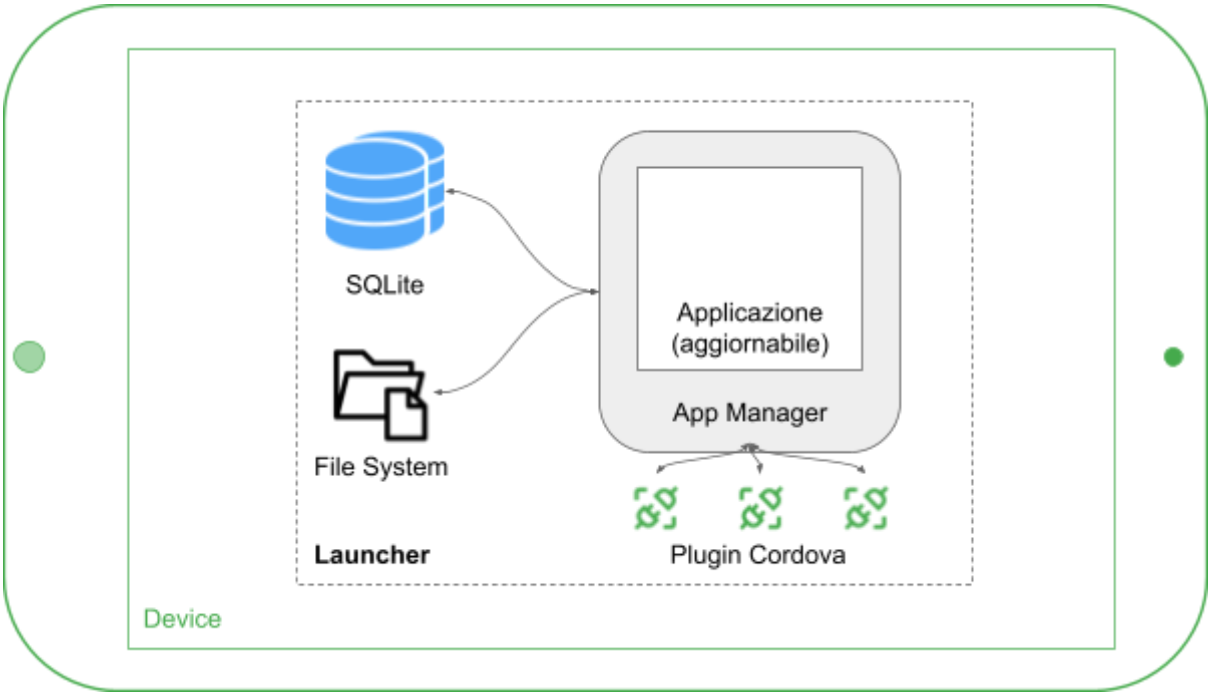
1. *web*: normali sessioni web che vengono istanziate tramite browser.
2. *rest*: sessioni che gestiscono un comando in modalità *rest*, solitamente proveniente da un sistema esterno che deve essere integrato.
3. *webapi*: sessioni che gestiscono una webapi di tipo OData generata con Instant Developer 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.

Questa architettura normalmente viene resa disponibile in un container Docker in esecuzione nella Google Cloud Platform in un data center europeo. In questo contesto:

1. È possibile installare più applicazioni nello stesso container. Ogni applicazione avrà un ambiente operativo separato.
2. Ogni applicazione accede ad un proprio file system separato dalle altre applicazioni.
3. Nel container è presente anche un database server Postgres che può contenere i database utilizzati dalle varie applicazioni installate nel container.
4. Per ogni applicazione è possibile configurare le politiche di gestione dei worker, assegnandoli ad un tipo di sessione o alla quantità delle stesse.

## Container per applicazioni mobile

Il container per applicazioni mobile di Instant Developer Cloud è chiamato *Launcher*, ed è basato sull'architettura Apache Cordova per la creazione di applicazioni mobile ibride.




Un launcher mette a disposizione dell'applicazione un device virtuale compatibile con quello del contenitore per applicazioni web: lo stesso codice applicativo viene eseguito con la medesima semantica, fatto salvo il diverso ambiente operativo su cui esso avviene, cioè una *webview* del dispositivo invece che un processo *node.js*.

Quando l'applicazione è in esecuzione nel cloud, le query sui database relazionali vengono eseguite rispetto al database Postgres. In questo caso invece le stesse query vengono eseguite sul database SQLite del dispositivo e il codice SQL viene automaticamente tradotto da Instant Developer Cloud, se la query è stata inserita in modo *strutturato*.

È disponibile un file system compatibile con quello del cloud e l'applicazione può accedere ai vari plugin Cordova tramite delle classi JavaScript di interfaccia. Nelle librerie standard sono già inclusi i plugin più usati, ma è possibile aggiungere i propri plugin e renderli disponibili all'applicazione sviluppando la relativa interfaccia.

L’applicazione vera e propria viene compilata dalla console attraverso un’operazione di build dell’applicazione sviluppata con l’IDE cloud. Sempre tramite la console è possibile ottenere il progetto Cordova che unisce l’applicazione e il launcher, oppure inviare direttamente l’applicazione compilata agli store.

L’applicazione è aggiornabile in tempo reale: dopo la prima installazione è possibile sostituire il codice applicativo in funzione del container senza dover nuovamente sottoporre l’applicazione agli store. Questa funzionalità viene gestita sempre tramite la console e permette di distribuire fix e piccoli miglioramenti che non richiedono cambiamenti alla struttura del launcher (plugin nativi, splash screen, dati di configurazione).

Per integrare l’applicazione mobile con il cloud è disponibile un framework di sincronizzazione che consente la remotizzazione delle operazioni sui dati e l’aggiornamento automatico dei dati nel database locale al dispositivo.

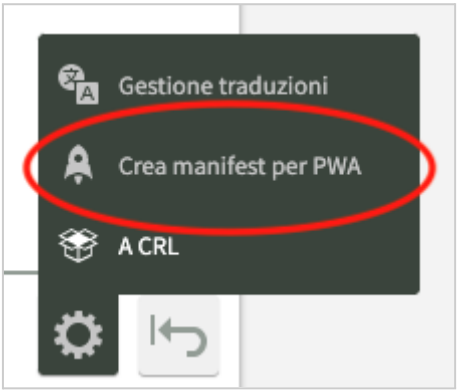
## Container per Progressive Web App (PWA)

Il container per applicazioni PWA di Instant Developer Cloud è molto simile a quello per le applicazioni mobile. Tuttavia in questo caso il device virtuale messo a disposizione delle applicazioni è basato solo sulle funzionalità del browser.

In questo senso il database SQLite è disponibile in alcuni contesti, ma non in tutti. La stessa cosa vale per il file system e per i vari plugin, la cui disponibilità può variare caso per caso.

Per maggiori informazioni sulla compatibilità del device virtuale nei vari contesti, si consiglia di leggere il documento: Manuale PWA.

Per ottenere una PWA occorre aggiungere all’applicazione le risorse necessarie tramite il comando *Crea manifest per PWA* del menu specifico dell’applicazione e poi attivare la modalità *offline* dell’installazione web nella console.




*Attivazione del manifest per PWA*

## Gerarchia delle classi di un’applicazione

Dopo aver analizzato il contesto in cui l’applicazione è mantenuta in funzione, cioè i vari tipi di application container disponibili, vediamo la struttura dell’applicazione stessa, che, sulla base di quanto illustrato in precedenza, sarà indipendente dal contesto di esecuzione.

L’architettura delle classi è definita tramite oggetti JavaScript basati su *prototype* ed ha la seguente struttura.


L’applicazione (*App*) è l’oggetto base della gerarchia e contiene tutte le definizioni delle videate, delle classi, dei database e delle librerie. Nel container per applicazioni web, *App* è un singleton per ogni worker e gestisce la lista delle sessioni in esecuzione in tale worker. Nei container per applicazioni mobile e PWA, *App* è un singleton che contiene l’unica sessione di lavoro in esecuzione nel device.

Le videate e le classi inserite come figlie dell’applicazione vengono definite all’interno di *App*. Se quindi si crea un videata di nome *MyView* all’interno dell’applicazione, essa potrà essere referenziata nel codice come *App.MyView*. La stessa cosa avviene per le classi definite allo stesso livello delle videate.

I database contengono gli schemi relazionali di cui l’applicazione fa uso; a livello di codice per ogni database viene definita una classe figlia di *App*. Se quindi si crea un database di nome *MyDB* all’interno del progetto, esso potrà essere referenziato come *App.MyDB* in ogni applicazione o libreria contenuta nel progetto stesso.

Le librerie contengono i documenti, le videate o le classi utilizzabili in ogni applicazione presente nel progetto. Per ogni libreria viene definita una classe figlia di *App*. Se quindi si vuole referenziare la classe *MyClass* contenuta in una libreria di nome *MyLib*, occorre scrivere *App.MyLib.MyClass*.

Ogni classe può contenere proprietà e metodi. Le proprietà sono sempre *di istanza*, quindi non è possibile definire nell'IDE proprietà statiche perché verrebbero condivise fra tutte le sessioni in esecuzione nel medesimo worker. I metodi, invece, possono essere sia *di istanza* che *statici.*

Una videata è un tipo di classe particolare che deriva da *Application.View* invece che da *Object*. Oltre alle proprietà e ai metodi, una videata contiene la gerarchia degli elementi visuali che ne costituiscono l'interfaccia utente. Sia la videata che i vari elementi possono contenere i gestori degli eventi notificati dalle azioni dell'utente o dal ciclo di vita della videata stessa.

Un database è un tipo di classe particolare che deriva da *Application.Database* invece che da *Object.* Un database non contiene codice, ma la definizione dello schema delle tabelle e delle relazioni fra di esse. Un database può contenere anche la definizione di liste valori da utilizzare come dominio dei campi. La classe base dei database mette a disposizione le funzioni per la gestione delle transazioni e per l'esecuzione di query in modo indipendente dal tipo di database.

La classe base dei database definiti nell'applicazione non è direttamente *Application. Database*, ma una più specifica, in funzione del tipo di database utilizzato. Sono disponibili *Application.Postgres*, *Application.SqlServer*, *Application.Oracle*, *Application.MySql* oppure *Application.Sqlite*. Da notare che quando si esegue la build dell'applicazione per un container mobile o PWA, tutti i database vengono temporaneamente considerati come derivanti da *Application.Sqlite*.

Un documento è un tipo di classe particolare che deriva da *Application.Document* invece che da Object. Il documento è l'entità base del framework ORM di Instant Developer e il suo schema può essere collegato ad una tabella definita in un database o ad una API OData importata nel progetto. Se viene collegata ad una tabella del database, la classe rappresenta la modalità standard di interazione con tale tipo di dati e una sua istanza rappresenta un record della tabella. Un documento si comporta in maniera analoga se viene collegato ad una API OData.

Sia l'applicazione che ogni classe contenuta in un progetto può contenere risorse di vario tipo, principalmente immagini, file di testo, json o css. Anche in questo caso è possibile referenziare le risorse nel codice applicativo indipendentemente dal contesto di esecuzione dell'applicazione.

# La programmazione relazionale

In questo paragrafo vogliamo descrivere la differenza strutturale più importante che distingue i processi di sviluppo con le piattaforme Instant Developer da quelli utilizzati in qualsiasi altro strumento di programmazione. Vogliamo inoltre analizzare i vantaggi che porta.

Normalmente lo sviluppo di software avviene memorizzando righe di codice in tanti file di testo. Questo metodo non rispetta la natura *relazionale* del software: ogni riga di codice contiene implicitamente delle dipendenze verso altre righe scritte in altri file, ma queste informazioni vengono perse e non sono disponibili in una forma trattabile dal sistema stesso.

Facciamo due esempi di quali relazioni implicite vengono perse durante la scrittura del codice. Nel primo esempio analizziamo una semplicissima funzione JavaScript:

```
function sommaInt (n1, n2) {
  let s = n1 + n2;
  return parseInt(s);
}
```

In questa funzione, viene persa l'informazione relativa ai tipi di parametri di input (numerici) e di output (numerico intero). Questa perdita è relativa al tipo di linguaggio JavaScript, che non è tipizzato staticamente, ed è superabile usando altri tipi di linguaggio più specifici.

Passiamo ad un esempio più complesso, ma presente in ogni progetto software che utilizza un database.

```
create table Prodotti (
  codice: integer not null,
  famiglia: varchar not null,
  nome: varchar not null,
  descrizione: varchar not null,
  primary key (codice)
}

async function getNomeProdotto (codiceProdotto) {
  let rs = await database.query(
    "select nome from prodotti where codice = " + parseInt(codiceProdotto));
  return rs.rows[0].nome;
}
```

Questo secondo esempio mostra le relazioni implicite fra il file che contiene la definizione dei dati del database e una funzione JavaScript che utilizza tali dati. Avendo scritto la query in una stringa, sono state implicitamente definite le seguenti relazioni:

- Il campo nome della tabella prodotti si chiama effettivamente nome.
- la tabella prodotti si chiama effettivamente prodotti.
- il campo codice della tabella prodotti si chiama effettivamente codice.
- la chiave primaria della tabella prodotti è costituita da un unico campo: codice.
- La sanificazione del campo di input (*codiceProdotto*) viene ottenuta tramite la funzione *parseInt* in quanto tale campo è numerico intero.
- Il tipo di ritorno della funzione è di tipo stringa.

La mancanza di una definizione esplicita di queste relazioni causa una rigidità del software che aumenta esponenzialmente al crescere complessità dello stesso.

Infatti, se diventasse necessario modificare il tipo di dati della colonna *codice*, oppure cambiare la chiave primaria della tabella aggiungendo ad esempio anche il campo *famiglia*, ci si dovrebbe ricordare di modificare manualmente la funzione *getNomeProdotto* per adattarla alle nuove specifiche. Ma questo adattamento può risultare in una modifica all'interfaccia della funzione e, in questo caso, diventerebbe necessaria una ulteriore fase di modifica manuale del software per verificare ed adattare tutti i punti in cui è stata usata quella funzione. E così via, in maniera ricorsiva, ogni modifica ad ogni riga di codice può rendere necessario analizzare manualmente gli impatti e correggere ulteriormente il codice.

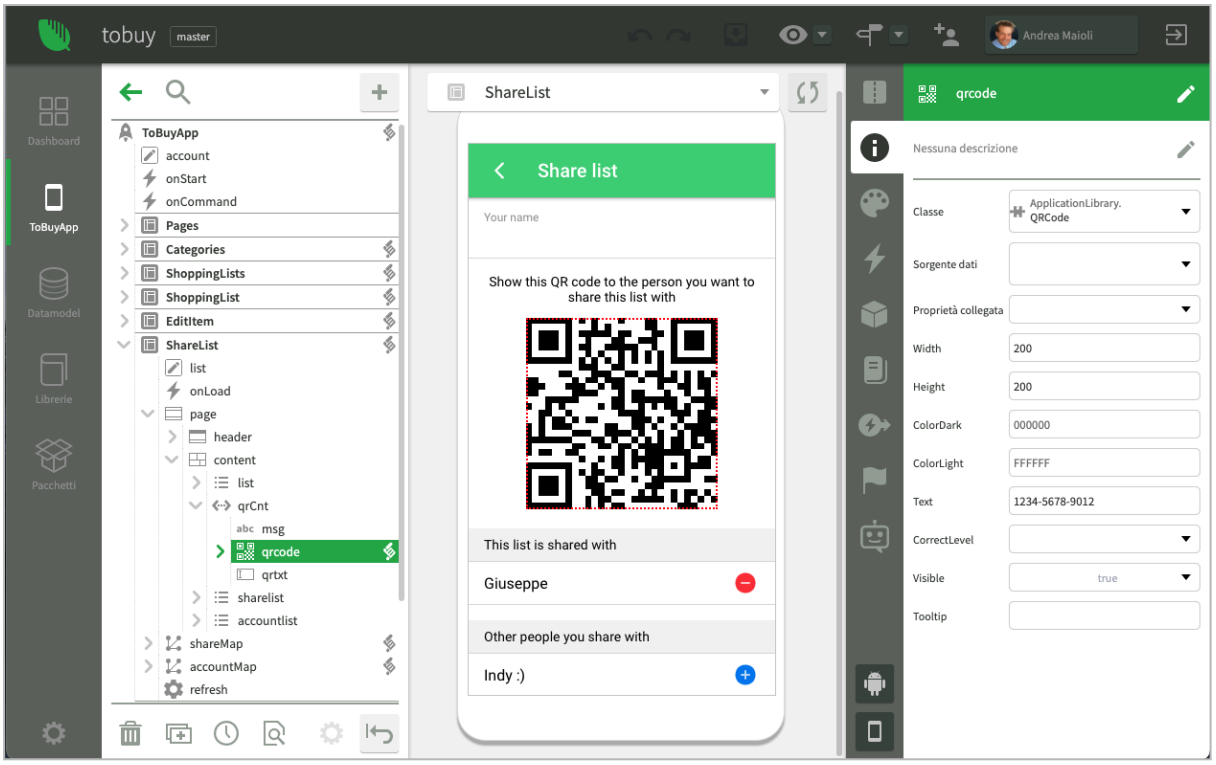
Questa è la causa del cosiddetto *debito tecnico*, ma aspetto ancora più negativo, è la causa dell'irrigidimento del software che nel tempo non è più in grado di essere modificato senza costi eccessivi. Il software, non potendo più evolvere per adattarsi alle mutevoli esigenze dell'ambiente, diventa inutile e deve essere completamente sostituito con un'operazione costosissima e spesso impossibile.

Come si può risolvere questo problema così importante? Utilizzando una nuova tecnologia di memorizzazione delle definizioni e del codice del progetto che tenga conto della natura relazionale descritta in precedenza e permetta al sistema di fornire le informazioni necessarie a comprendere l'impatto di una modifica, o, meglio ancora, ad eseguirla automaticamente. Questa tecnologia deve tenere conto contemporaneamente di tutti gli aspetti del progetto software, non si può limitare a gestire i vari pezzi in maniera scollegata: la struttura del database non può essere separata dal codice applicativo che lo utilizza.

Ecco perché Instant Developer Cloud memorizza i dati di un progetto software non in tanti file di testo, ma utilizzando un grafo memorizzato in un unico -grande- oggetto JavaScript avente come nodi tutti gli oggetti definiti nel progetto, e come archi le relazioni fra di essi.

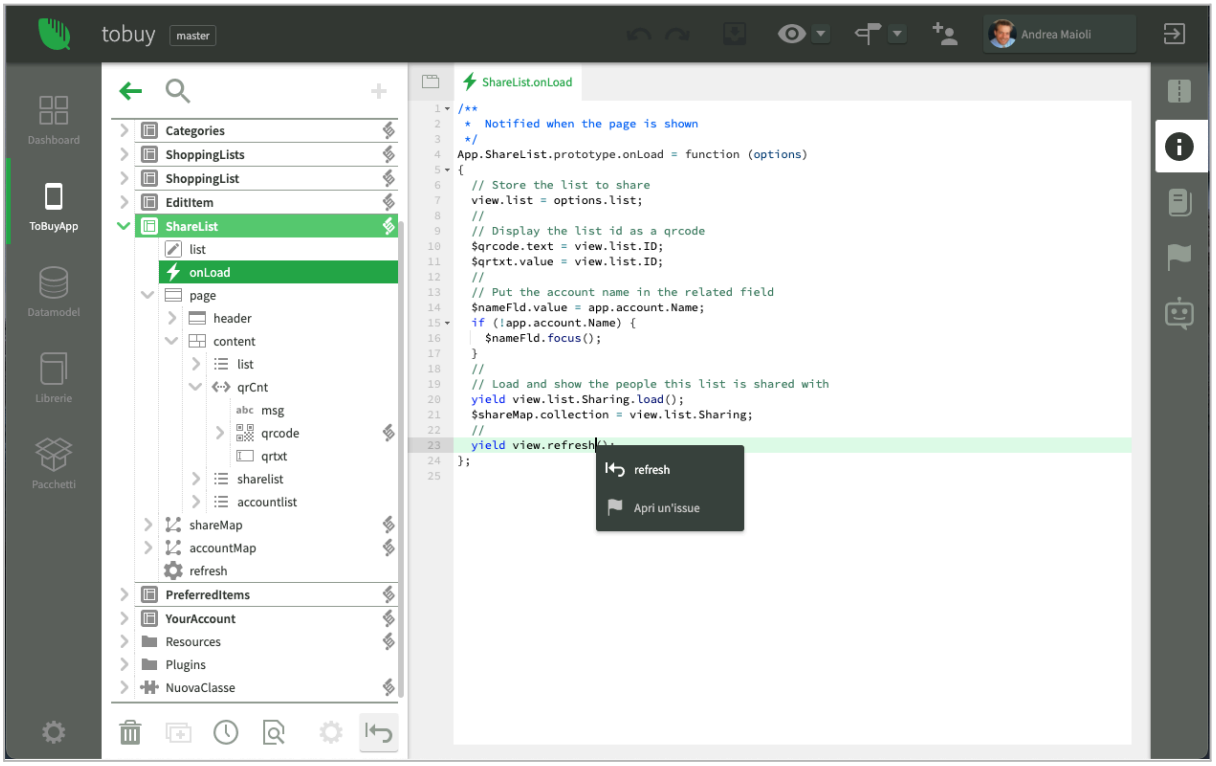
Quali sono le parti di progetto memorizzate nei nodi di questo grafo? In sostanza tutto ciò che è presente, che in qualche modo è stato definito. Facciamo alcuni esempi:

- Le videate dell'applicazione
- Gli elementi grafici che compongono le videate.
- Gli script che gestiscono gli eventi degli elementi grafici.
- Le classi.
- Le proprietà ed i metodi delle classi.
- I database, le tabelle, i campi, le relazioni e gli indici.
- I documenti e i relativi metodi, proprietà ed eventi.
- Le librerie di sistema.
- I pacchetti.
- Le risorse (immagini, video, file…).




*L'albero del progetto mostra una parte dei nodi del grafo che descrive il sistema in fase di sviluppo*

Oltre a tutti questi elementi, anche tutto il codice presente in ogni script dell'applicazione viene memorizzato come parte del grafo. Quando selezioniamo nell'albero degli oggetti uno script o un metodo, l'editor di codice ricostruisce una versione testuale della parte di grafo dedicata al metodo e la mostra in un editor di codice JavaScript.




*Anche il codice è parte del grafo del progetto*

Quando il codice viene modificato, il grafo sottostante viene ricostruito in tempo reale, in modo tale che tutto quello che viene scritto nell'IDE rimanga sempre in forma relazionata.

Nell'immagine precedente possiamo vedere che all'interno del nodo *onLoad*, che rappresenta lo script lanciato all'apertura di una istanza della videata *ShareList*, vengono memorizzati i seguenti tipi di nodi:

- I parametri del metodo, ognuno con le proprie caratteristiche.
- Le righe di codice del metodo
- Per ogni riga, ogni token o parola chiave del linguaggio.

Ogni nodo del grafo ha una serie di caratteristiche che dipendono dal tipo di nodo stesso. Ad esempio, un elemento grafico memorizza tutte le proprietà dell'elemento stesso, che a sua volta dipendono dal tipo di elemento in funzione di come esso è stato definito nelle librerie.

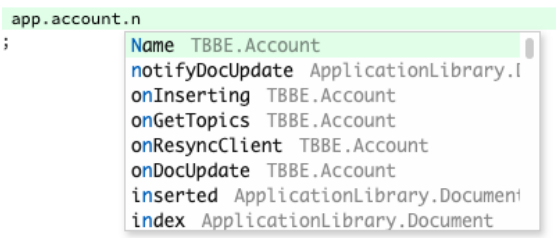
I nodi che rappresentano le righe e i token di codice portano importantissime informazioni di collegamento con gli oggetti presenti nel progetto. In questo modo, infatti è possibile far funzionare gli algoritmi di type inference e di gestione delle referenze. Come esempio di questo possiamo notare nell'immagine precedente che aprendo un menu contestuale a partire dal token *refresh* appare una voce di menu che permette di saltare alla definizione del metodo nella videata. Questa operazione non avviene tramite matching di stringhe, ma proprio come referenza diretta del token al nodo *metodo refresh* nell'albero degli oggetti.

## Vantaggi del modello relazionale

La modalità di memorizzazione relazionale rende possibili alcune delle caratteristiche più interessanti e utili di Instant Developer Cloud fra le quali:

*Instant refactoring*: modificando il nome o le caratteristiche di un elemento del progetto, le conseguenze di questa modifica vengono apportate automaticamente in tutto il codice presente nel progetto. Se, ad esempio, si cambia il nome al metodo *refresh* in *aggiorna*, anche le righe di codice che lo chiamavano vengono cambiate di conseguenza. Tali modifiche vengono apportate in automatico perchè sono sicure, non richiedono alcuna revisione manuale.

*Type inference*: Instant Developer Cloud è in grado di recuperare in tempo reale i tipi riferiti dalla maggior parte delle espressioni JavaScript, anche se questo linguaggio, per sua natura, non ha alcuna informazione sui tipi a design time. In questo modo è possibile proporre un *intellisense* globale esteso all'intero progetto: dal database, alle librerie di componenti, fino ad ogni elemento di back-end e front-end.




*Intellisense globale sull'intero progetto*

*Sequenzializzatore asincrono*: una delle difficoltà maggiori della scrittura di applicazioni JavaScript complesse risiede nella gestione delle operazioni asincrone, che, per loro natura, richiedono di descrivere il flusso di un processo complesso tramite callback, come mostrato nell'esempio seguente.

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

*L'inferno delle callback*

Questo modo di scrivere il codice viene spesso chiamato l'inferno delle callback perché rende estremamente difficile riconoscere il significato del codice, oltre che essere altamente imprevedibile per quanto riguarda la gestione delle eccezioni. Cosa accade infatti in caso di eccezione JavaScript? Quale sarà la prossima riga di codice ad essere eseguita?

Fin dal 2015 Instant Developer Cloud contiene un framework per la gestione automatica di questo problema. Tale framework converte il codice asincrono in un codice sequenzializzato tramite *yield*. Le righe di codice che coinvolgono operazioni asincrone vengono quindi *sospese* dal costrutto *yield* e poi riprese in automatico al termine dell'operazione asincrona. Anche la gestione delle eccezioni può essere scritta in modo normale, senza preoccuparsi del livello di asincronicità in cui esse avvengono.

Per poter funzionare, l'algoritmo di sequenzializzazione asincrona ha bisogno di informazioni dettagliate sui tipi di oggetti utilizzati nel codice, in quanto il codice stesso deve cambiare di forma se vengono coinvolte o meno operazioni asincrone. Quindi un metodo che inizialmente non è asincrono e lo diventa in seguito, causa modifiche automatiche anche in tutti i punti in cui tale metodo è stato utilizzato.

*Gestione globale delle referenze*: quando il progetto software diventa complesso, è necessario poter rilevare dove un determinato oggetto viene utilizzato. Ad esempio si potrebbe essere interessati a conoscere tutti gli utilizzi di un determinato campo (ad esempio di nome "id") di una tabella del database. Instant Developer Cloud è in grado di estrarre queste informazioni in modo rapido e sicuro navigando il grafo dei nodi del progetto.

## Effetti del modello relazionale

L'esperienza di programmazione con Instant Developer Cloud offre quindi il vantaggio di "tenere sempre in mano" un progetto, anche di classe enterprise, con un utilizzo minimo di risorse (tempo e costi). Questo vale nella fase iniziale di sviluppo, e vale ancora di più nella fase di manutenzione nel tempo, quando è più facile perdere conoscenza delle relazioni fra i vari elementi del software.

# Apprendere l'uso di Instant Developer Cloud

## Prerequisiti

Per poter utilizzare con successo Instant Developer Cloud è necessario avere familiarità con i seguenti argomenti:

- Database relazionali.
- Programmazione Object-Oriented.
- Linguaggio JavaScript.
- HTML e CSS.

Se si desidera approfondire uno di questi argomenti, un metodo efficace può essere quello di frequentare un corso online, ad esempio uno dei seguenti:

- CodeAcademy - Learn SQL
- Udemy: Object-oriented Programming in JavaScript
- Code Academy: Learn JavaScript
- Code Academy: Learn HTML
- Code Academy: Learn CSS

## Strumenti di autoformazione

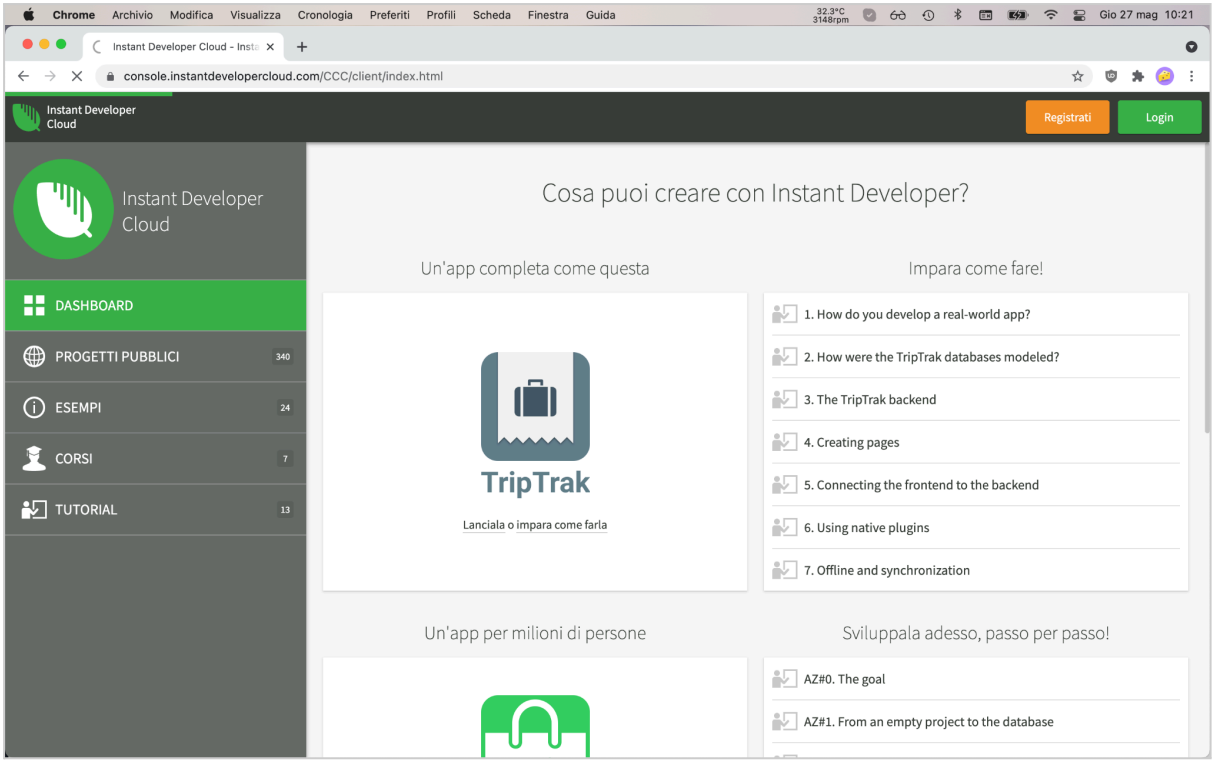
Instant Developer Cloud contiene numerosi strumenti di autoformazione che rappresentano un passo essenziale di un percorso formativo di successo.

### Tutorial

Il sistema dei tutorial è accessibile dalla console di Instant Developer Cloud anche senza aver effettuato la registrazione o il login. È sufficiente collegarsi con la console all'indirizzo console.instantdevelopercloud.com per accedere al sistema dei tutorial, come mostrato nell'immagine alla pagina seguente.

I tutorial sono organizzati in due sezioni rispettivamente di sette e sei lezioni ciascuna. Ogni tutorial avvia una sessione IDE in modalità guidata e l'avanzamento della lezione avviene tramite un sistema di chat interattivo. Il tempo necessario per ogni lezione è di circa 10-15 minuti.

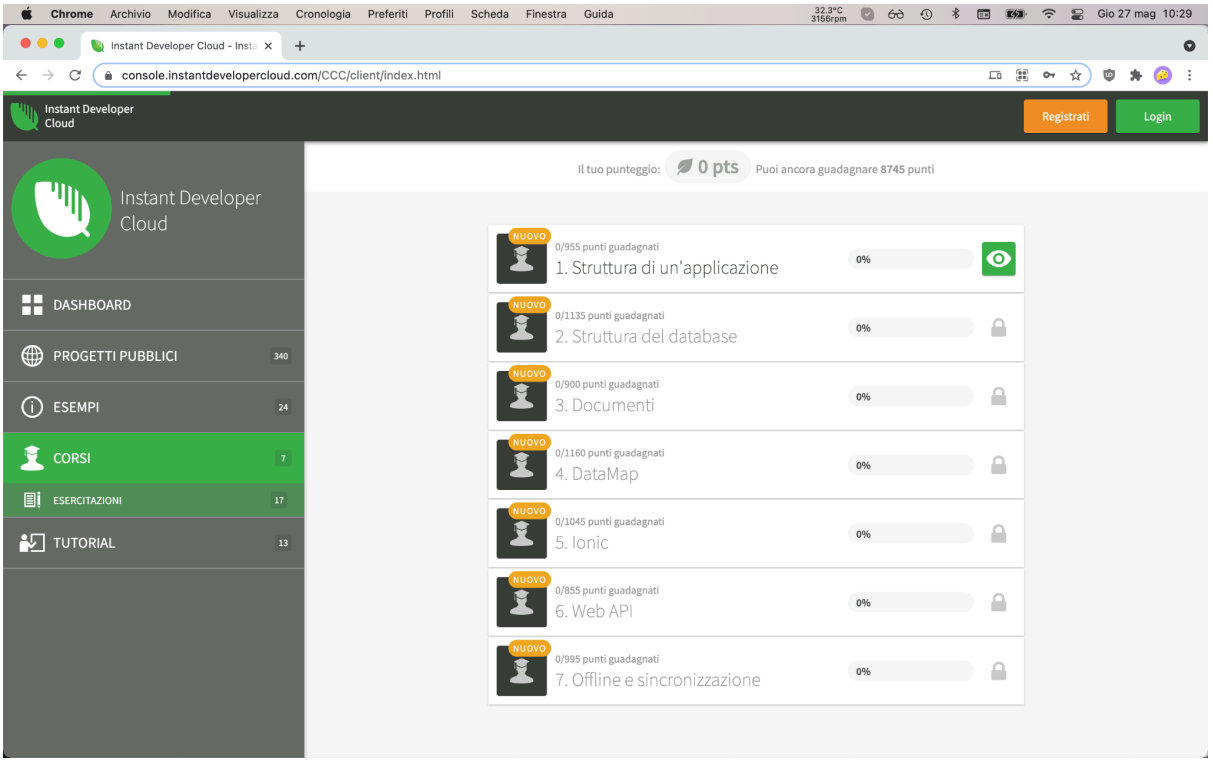
Lo scopo dei tutorial è quello di prendere confidenza con l'IDE di Instant Developer Cloud senza avere alcun tipo di conoscenza pregressa. I tutorial presentano i meccanismi base del funzionamento di Instant Developer Cloud in due casi d'uso reali.




*Accedere al sistema dei tutorial*

## Corsi “Academy”

Il percorso formativo vero e proprio comincia con i corsi *academy*, disponibili dopo aver effettuato la registrazione gratuita alla piattaforma.




*Corsi ed esercitazioni*

Il percorso di autoformazione si compone di sette moduli, ognuno dei quali contiene dalle sei alle otto lezioni. Le lezioni avvengono nell'IDE in modalità libera; ogni lezione si compone di un determinato numero di task da eseguire per arrivare all'obiettivo e il sistema è in grado di controllare l'esattezza di ogni passaggio. Il tempo necessario per completare una lezione è compreso tra 15 e 30 minuti e sono necessari i prerequisiti indicati in precedenza per poter completare con successo il percorso.

Sono inoltre disponibili 16 esercitazioni che consentono di verificare le conoscenze acquisite e di sfidare se stessi nell'ottenimento di risultati applicativamente significativi. Le esercitazioni avvengono nell'IDE in modalità completamente libera e viene verificato solo l'ottenimento del risultato corretto, che viene registrato nella console. Completando con successo una esercitazione verranno sbloccate le successive e si potrà vedere la soluzione proposta di quelle precedenti.

### Esempi

Nella console sono presenti anche una serie di esempi consultabili anche senza aver effettuato la registrazione.

Gli esempi sono suddivisi in tre sezioni:

- *Applicazioni complete*: esempi di applicazioni con funzionalità complete testabili sia in modalità installata che nell'IDE di sviluppo.
- *Design pattern*: esempi relativi alla realizzazione di particolari funzioni applicative. Si consiglia di consultare i design pattern dall'IDE di sviluppo in modo da poter visualizzare il codice relativo. Alcuni esempi di questo tipo sono testabili anche in modalità installata.
- *Demo di componenti*: esempi di uso di particolari componenti di interfaccia utente o librerie di sistema.

Gli esempi sono collegati alla documentazione in linea consultabile dall'IDE in modo da poter avere un'evidenza immediata di utilizzo di una determinata funzione di libreria.

## Apprendimento guidato: lo Starter Kit

Seguendo il percorso di autoformazione illustrato in precedenza, viene fatta esperienza dei meccanismi più importanti della fase di sviluppo con Instant Developer Cloud.

Tuttavia, quando si tratta di sviluppare un progetto con tempi e modalità definite, è importante avere la certezza di seguire il percorso di implementazione corretto fin dal primo tentativo.

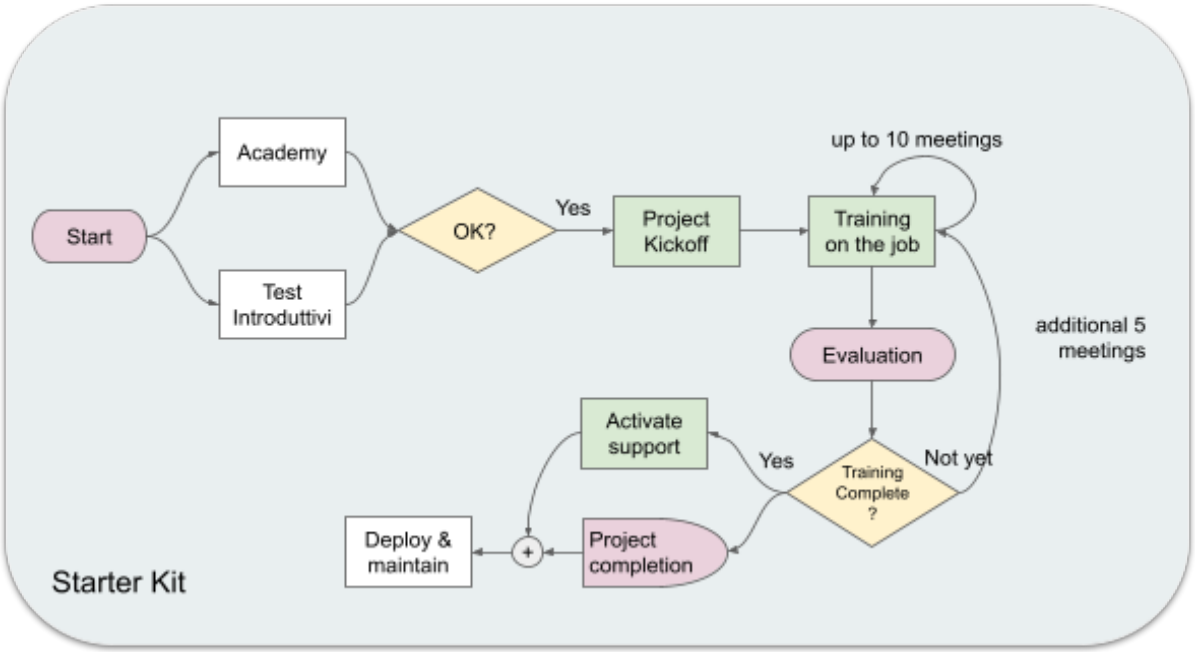
Per rispondere a questa esigenza, il team di Instant Developer mette a disposizione un percorso formativo guidato chiamato *Starter Kit*, che permette di seguire i passaggi necessari allo sviluppo coerente di un proprio sistema omnichannel.

Lo Starter Kit è acquistabile tramite console e comprende tutte le risorse necessarie al suo completamento, in particolare:

- Le licenze d'uso di Instant Developer Cloud
- L'utilizzo di un server di produzione (Server App IDC) per tre mesi
- L'utilizzo di un launcher per tre mesi.
- La riunione di pianificazione e la correzione dei test introduttivi.
- Le prime dieci sessioni di formazione con tutor.

Il tempo necessario per completare uno Starter Kit è di circa 2-3 mesi, comprendenti anche il tempo per le attività di autoformazione indicate sopra. Per maggiori informazioni sugli Starter Kit, è possibile inviare una mail a commerciale@instantdeveloper.com.

## Funzionamento dello Starter Kit




*Schema di funzionamento di uno Starter Kit*

Lo Starter Kit inizia con due attività da svolgere in parallelo:

- *Test introduttivi*: svolgimento di un di test di ingresso per la verifica dei prerequisiti del programmatore in formazione: database e SQL, programmazione Object-Oriented, programmazione JavaScript, HTML e CSS. Nel caso in cui i test non abbiano esito positivo si consiglierà il rafforzamento delle conoscenze di base tramite l'esecuzione di corsi online.
- Svolgimento delle attività di autoformazione descritte al paragrafo precedente: tutorial, academy ed esercitazioni.

Il passo successivo è la riunione di *Kick Off* del progetto nella quale il tutor, il project manager e il programmatore verificano il progetto da realizzare e pianificano il calendario delle riunioni di training. Nella riunione di *Kick Off* del progetto si validano:

- La struttura del database.
- L'analisi di massima dell'applicazione da realizzare.
- Il mockup o wireframe delle videate dell'applicazione.
- L'architettura tecnologica a supporto.

A questo punto inizia il percorso di formazione guidata vero e proprio: le riunioni di *Training on the job* si svolgono a cadenza settimanale e hanno una durata di un'ora. Durante queste riunioni il tutor verifica il lavoro svolto dallo sviluppatore, risponde a domande specifiche emerse durante lo sviluppo, aiuta ad imbastire la struttura corretta delle varie parti del progetto e assegna i compiti da svolgere per la riunione seguente.

Al termine delle dieci riunioni di *training* viene eseguita una riunione di valutazione dello stato del progetto di formazione (*Evaluation*), in particolare si valuta il grado di conoscenza di Instant Developer Cloud raggiunto dal programmatore. Nel caso in cui la formazione non sia ritenuta sufficiente si possono aggiungere ulteriori riunioni di *training* per colmare le lacune rimaste.

Quando il grado di formazione è sufficiente, il processo si conclude con l'attivazione del servizio di formazione continua fornito dal supporto di Pro Gamma. In questa fase viene pianificata una riunione con il responsabile del supporto dove verranno illustrate le procedure per richiedere assistenza al corretto utilizzo, consulenza, ma anche come segnalare eventuali malfunzionamenti o errori di Instant Developer Cloud.

A questo punto il processo Starter Kit è concluso (*Project completion*) in quanto si è in grado di procedere con l'utilizzo della piattaforma in autonomia al fine di installare, gestire e manutenere le proprie applicazioni (*Deploy & maintain*).

## Formazione continua e assistenza al corretto utilizzo

Al termine del percorso di autoformazione o dello Starter Kit, se si è in possesso di una licenza d'uso di Instant Developer Cloud è possibile accedere ai servizi di formazione continua forniti dal team di Instant Developer.

Essi sono mirati ad accompagnare gli sviluppatori e i project manager in tutte le fasi del ciclo di vita delle proprie applicazioni. Si suddividono in due tipologie: assistenza al corretto utilizzo e consulenza vera e propria.

**Assistenza al corretto utilizzo**: è un servizio di affiancamento fornito tramite sistemi di telecollaborazione che aiuta il richiedente ad affrontare un problema nell'utilizzo dei prodotti Instant Developer fornendo consigli, spiegazioni o controllando il codice del progetto.

**Consulenza**: è un servizio di assistenza mirato ad affrontare un problema complesso e a fornire una risposta strutturata. Può essere usato per richiedere una consulenza architetturale sia sull'infrastruttura che sull'organizzazione del progetto o per integrare particolari dispositivi o servizi di terze parti.

È possibile acquistare e richiedere i servizi di formazione continua tramite l'apposito modulo nella console. Per maggiori informazioni è possibile inviare una mail a commerciale@instantdeveloper.com.