Debugging e test

Indice generale

Debugging e test

Impara le tecniche di debug migliori. Imposta i test automatici di non regressione. Ottimizza il codice con i test di carico.

Introduzione

Lo sviluppo del software è un'attività costantemente sottoposta a verifica. Non appena un programmatore scrive dieci righe di codice, subito desidera effettuare un test per vedere se il programma si comporta come previsto.

Avere buoni strumenti di debugging integrati con l'ambiente di sviluppo è quindi una necessità importante. Inoltre le necessità di verifica non si esauriscono nella fase di sviluppo: prima di installare, è necessario verificare che una nuova versione funzioni ancora correttamente. È quindi necessario un sistema di test automatico per scoprire le eventuali regressioni introdotte dal nuovo codice.

Le verifiche non si fermano al funzionamento corretto dell'applicazione. Oggigiorno occorre ottimizzare il consumo di risorse, in quanto si devono gestire carichi di lavoro importanti, ed è necessario che l'applicazione sia facilmente scalabile. Ecco perché avere un sistema di test di carico diventa una necessità.

Infine quando l'applicazione è già in produzione, in mano agli utenti finali, le anomalie di funzionamento diventano più difficili da rilevare, ma anche più urgenti. È quindi necessario un sistema che permetta di raccogliere le informazioni sullo stato dell'applicazione mentre essa è in esecuzione, sessione per sessione. Fino ad arrivare al debugging in tempo reale della sessione utente che causa problemi, mentre essa è ancora in esecuzione.

La piattaforma Instant Developer Cloud risolve tutte queste problematiche. Infatti possiede:

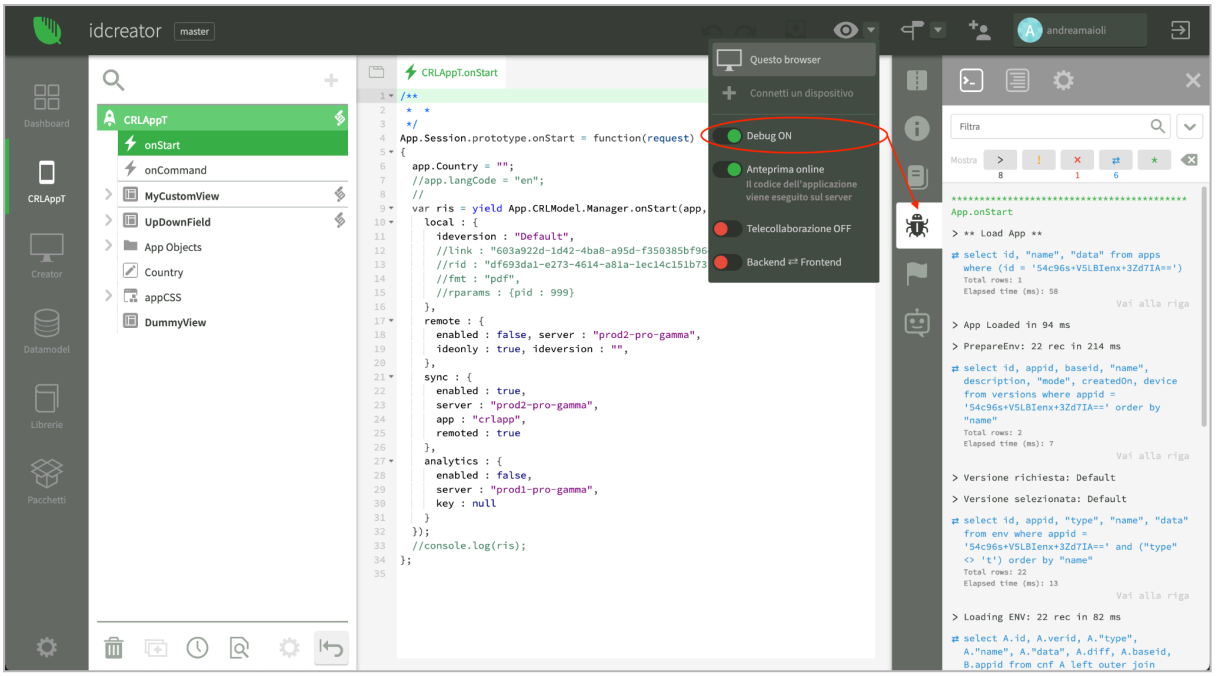
1) Strumenti di debug integrati nell'IDE, che comprendono anche l'analisi dell'impatto dell'applicazione sul database.
2) Un sistema per la realizzazione di test automatici, basati sulla registrazione e riproduzione di sessioni campione, che può essere usato per le verifiche di non regressione e per i test di carico.
3) Un sistema di raccolta dati analitici e tecnici dalle sessioni in produzione, sia nei device mobile, anche quando offline, che nei server di produzione.
4) Un sistema di debugging in tempo reale delle sessioni utente nei server di produzione mentre esse sono in esecuzione.

I sistemi di debug e test di Instant Developer Cloud sono testati sul campo per consentire un'esperienza di programmazione semplice e veloce. Vediamo nel dettaglio come funzionano.

Strumenti e tecniche di debug

Normalmente le prime sessioni di debugging del codice avvengono nell'IDE, lanciando l'applicazione in anteprima. Per attivare il sistema di debug occorre che il flag corrispondente contenuto nelle proprietà di attivazione dell'anteprima sia attivo, come è per default.

In questo caso, mentre l'applicazione è in funzione nell'anteprima, il pannello di debug contenuto nella barra destra degli strumenti mostrerà le informazioni raccolte dall'applicazione.



Il pannello di debug è suddiviso in tre sezioni, selezionabili con le icone contenute nell'intestazione: *console*, *trace* e *impostazioni*.

La sezione console

La sezione *console* contiene l'output delle funzioni di console (*console.log*, *console.warn, console.error* e *console.trace*), sia quelle inserite dall'utente nel proprio codice che quelle contenute nel framework. In questa sezione vengono stampate anche le query eseguite dall'applicazione in modo da rendersi conto sia della qualità che della quantità di accessi al database che l'applicazione esegue in base agli input dell'utente.

Tramite la sezione filtro in alto è possibile filtrare il contenuto della console, ma anche selezionare solo alcuni tipi di informazioni tramite i pulsanti posti sotto al campo di filtro. È possibile inoltre cancellare il contenuto della console con l'ultimo pulsante a destra dei filtri.

Al di sotto dei pulsanti di filtro è possibile sapere quante ricorrenze sono presenti nella console dall'inizio della sessione. In questo modo è possibile un'informazione sintetica su quante query sono state eseguite nella sessione in fase di test.

Le informazioni contenute nella console sono raggruppate in *richieste*, corrispondenti alle intestazioni di colore verde. Ogni richiesta inizia da un determinato input inviato all'applicazione, come ad esempio la pressione di un pulsante da parte dell'utente, e contiene i messaggi generati dalla gestione di questo input da parte dell'applicazione. Cliccando sull'intestazione della richiesta è possibile collassare o espandere il contenuto.

Se l'applicazione genera un numero molto elevato di messaggi di log in un determinato intervallo di tempo, il sistema di debug ne interrompe temporaneamente la raccolta per evitare di sovraccaricare l'IDE.

La console è il principale strumento di debug del codice, in quanto consente di inserire nei punti opportuni i messaggi e la stampa dei valori delle variabili di interesse. Si noti che è possibile aggiungere o togliere istruzioni di console dal codice dell'applicazione durante la sessione di debug **senza dover riavviare** l'applicazione.

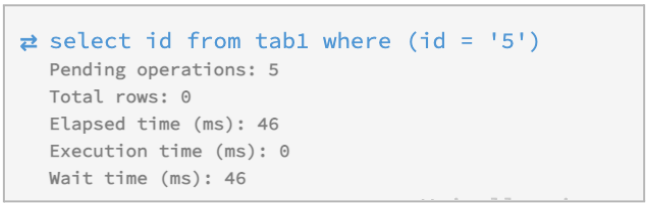
Allo stesso modo, tutte le modifiche al codice vengono applicate immediatamente. È quindi possibile aggiustare il codice e riprovare senza perdere tempo a far ripartire l'anteprima.

Cliccando su una riga relativa alla console si aprirà il codice che l'ha generata. Cliccando su una query, invece, si apriranno il query editor con il testo completo e un'anteprima delle prime dieci righe del risultato ottenuto. Se un messaggio o una query è stata eseguita da un processo di framework, l'IDE mostrerà la riga di codice *più vicina* a quella che ha causato l'emissione del messaggio.

Per le query e gli errori, infine, è presente il link *Vai alla riga* che permette di saltare alla riga di codice che ha generato la query o l'errore o a quella più vicina se il messaggio è stato generato dal framework.

Si noti infine che per ogni query vengono riportate informazioni aggiuntive, come ad esempio, il tempo di esecuzione o il numero di righe restituite. In alcuni casi vengono riportati anche il numero di operazioni pendenti (*pending operations*) e il tempo di attesa (*wait time*).

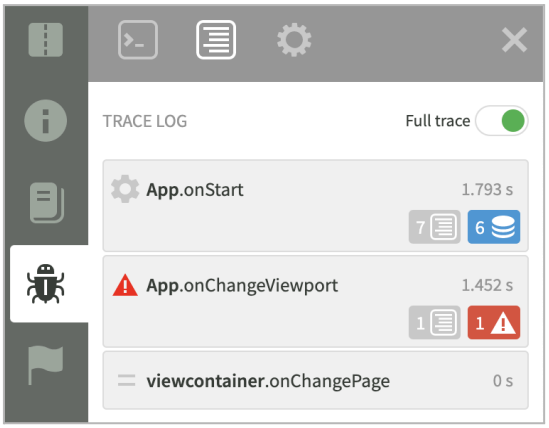
Queste informazioni sono particolarmente importanti perché indicano che la query è stata inviata al database mentre la connessione ne stava già processando altre per la stessa sessione di lavoro. In questi casi la query dovrà aspettare che la transazione precedente si concluda, ed è importante valutare che il tempo di attesa potrebbe essere evitato organizzando un diverso metodo di accesso al database.



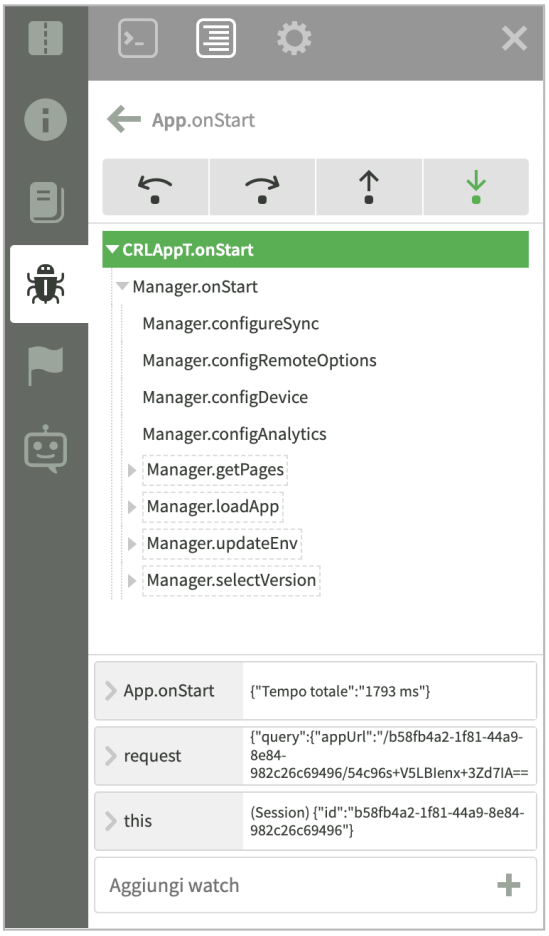
Esempio di informazioni aggiuntive riportate per una query

La sezione trace

La sezione *trace* contiene la lista delle richieste arrivate all'applicazione e le relative tracce di codice eseguite. Per ogni richiesta vengono indicati il relativo tempo di esecuzione e il numero di messaggi di log, warning, eccezioni e query eseguite durante la gestione della richiesta. Questo aiuta ad identificare le richieste particolarmente pesanti sul database.



Per attivare l'analisi completa delle tracce di codice, occorre attivare il flag *Full trace*. A questo punto sarà possibile vedere l'elenco dei metodi eseguiti da una determinata azione. Nell'immagine seguente vediamo un esempio di analisi delle tracce di codice relative ad una richiesta dell'applicazione.



Cliccando su un metodo, esso si aprirà nell'editor di codice nella parte centrale; inoltre sarà possibile seguire il codice passo passo con i pulsanti di navigazione posti nella parte alta della videata. Ad ogni passo, la riga di codice corrispondente verrà evidenziata in giallo nell'editor e nella parte inferiore del pannello di *trace* verranno mostrati i valori di tutte le variabili interessate dal codice eseguito.

A causa dell'esecuzione asincrona del codice sia in ambiente *Node.js* che in ambiente iOS e Android, è possibile che, procedendo alla prossima riga di codice, il flusso di codice passi in un altro metodo che viene eseguito mentre l'istruzione precedente è in attesa del completamento. Usando i pulsanti di navigazione sulla destra (*Step into* e *Back step into*) si seguirà l'esecuzione effettiva del codice, tenendo conto delle operazioni asincrone. Se invece si è interessati solo al metodo corrente, si possono utilizzare i pulsanti di navigazione sulla sinistra (*Step over* e *Back step over*) che limitano la navigazione al metodo corrente.

Utilizzando il comando *Vai alla riga* nel menu contestuale dell'editor di codice, è possibile posizionare il modulo di debug direttamente in quel punto e poi iniziare la navigazione del codice. Se la riga non è presente nella traccia di codice, verrà mostrato un messaggio.

È possibile inserire *breakpoint* cliccando nell'editor sul numero della riga di codice. In realtà il codice non viene veramente interrotto, tuttavia la sezione *trace* si posiziona immediatamente sul contesto in cui il *breakpoint* è stato raggiunto.

Oltre ai valori delle variabili mostrate nel pannello di trace, è possibile aggiungere ulteriori espressioni (*watch*) al metodo corrente; tali espressioni vengono valutate ogni volta che viene eseguita una riga del metodo a cui appartiene il watch. Dopo aver aggiunto un *watch* è necessario inviare una nuova richiesta all'applicazione in anteprima per vedere i dati raccolti.

Si fa presente che per evitare di raccogliere una traccia di codice troppo lunga o complessa, la raccolta dei dati viene temporaneamente disabilitata quando si entra in un ciclo *for*, *while* oppure *do,* dopo la decima ricorrenza del ciclo. In questo caso viene visualizzato un messaggio nel pannello di *trace*. Se in determinati casi è necessario tracciare un numero di ricorrenze maggiori, prima di eseguire il ciclo è possibile impostare la proprietà *App.DTT.DebugCycles* al numero desiderato, ad esempio 20.

La sezione impostazioni

Nella sezione *impostazioni* è possibile gestire l'elenco dei watch, l'elenco dei breakpoint e l'elenco degli script esclusi.

Uno script escluso non genera informazioni di *trace*, anche se il risultato dei metodi di console e delle query verrà comunque raccolto. Questo può essere importante in due casi:

- Lo script viene eseguito un numero considerevole di volte e i dati di trace da esso generati non sono significativi.
- Il codice generato dal sistema di debug interferisce con il comportamento dello script, e quindi deve essere eliminato.

Per aggiungere uno script alla lista degli esclusi è possibile effettuare il drag & drop dall'albero del progetto verso la lista degli script esclusi.

I watch

Il sistema di debug consente di inserire watch, cioè espressioni che vengono valutate per ogni riga di codice eseguita all'interno di un metodo. Al cambiare del valore dell'espressione, il nuovo valore viene inserito nella sezione *console* e mostrato nel *trace*.

I watch hanno il vantaggio di poter essere inseriti nell'IDE senza modificare il codice dell'applicazione, inoltre possono essere aggiunti, eliminati e modificati senza dover riavviare l'anteprima. Tuttavia, è possibile eseguire il watch solo di espressioni semplici, senza calcoli o chiamate a metodi.

Un'alternativa all'uso dei watch è l'inserimento di *console.log* nel codice, potendo così tracciare qualunque cosa. Tuttavia si ricorda che i *console.log* vengono eseguiti anche nel server di produzione ed entrano a far parte del log strutturato dell'applicazione. Tutti i console.log relativi ad una specifica sessione di debug dovranno quindi essere rimossi prima di installare l'applicazione.

Per inserire un watch, selezionare nell'editor di codice l'espressione della proprietà da controllare e selezionare la voce *Aggiungi watch* del menu contestuale nell'editor.

Proprietà modificabili del modulo di debug

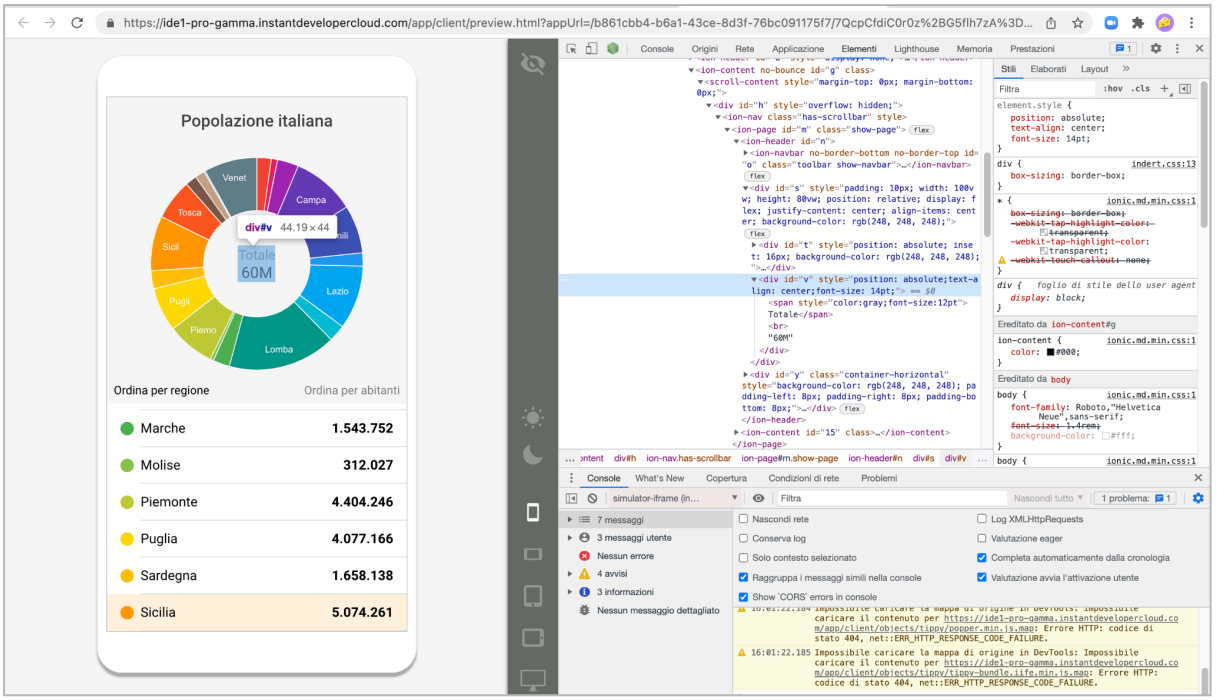
Oltre alla proprietà *DebugCycles* vista prima, il modulo di raccolta dati di debug contiene diverse proprietà che ne configurano il comportamento. Si consiglia di modificarne il valore solo per gestire casi eccezionali in quanto un valore errato può compromettere la stabilità dell'applicazione in anteprima e la responsività dell'IDE. La lista è la seguente:

- *App.DTT.QueryRows*: massimo numero di righe risultato di una query loggata dal modulo di debug. Default 10.
- *App.DTT.MaxItems:* massimo numero di item (righe di codice eseguite, messaggi, eccetera) raccolti in una singola richiesta. Default 50.000.
- *App.DTT.DebugCycles:* massimo numero di ricorrenze di un loop loggate. Default 10.
- *App.DTT.ArrayLogLimit*: massimo numero di item loggati di un array. Default 100.
- *App.DTT.StringLogLimit*: lunghezza massima di una stringa loggata dal sistema di debug prima del troncamento. Default 5000.
- *App.DTT.WatchDogLimit*: timeout in millisecondi prima di rilevare codice bloccato. Default 10000.

Debug della costruzione della videata

Vediamo ora cosa fare quando l'aspetto della videata non è quello che si prevede: questi problemi di solito riguardano la struttura degli elementi visuali o le impostazioni di stile.

Lo strumento migliore per gestire questa classe di problemi è il debugger integrato nel browser Chrome. Tramite i suoi strumenti è possibile verificare la struttura degli elementi e gli stili applicati, inoltre è possibile modificare tali stili in tempo reale fino ad ottenere i risultati desiderati. A questo punto è possibile correggere gli stili applicati nell'IDE.



Si ricorda che modificando gli stili, le proprietà degli oggetti visuali, oppure facendo modifiche ai file CSS o al codice mentre l'anteprima dell'applicazione è aperta, tali modifiche verranno applicate immediatamente senza dover riavviare.

Per correggere eventuali problemi di visualizzazione in un device, è possibile copiare il link dell'anteprima e incollarlo nel browser nativo del dispositivo. A questo punto l'anteprima apparirà anche nel browser e, se il dispositivo è collegato al proprio computer, si potrà effettuare il debug della pagina con gli strumenti di Chrome se il dispositivo è Android, o di Safari nel caso iOS. Su internet sono disponibili diverse guide su come collegare e configurare i propri dispositivi per consentire il debug delle pagine web. Ai seguenti link potete trovare un esempio per Android o per iOS.

Debug di applicazioni nei device

Dopo aver verificato che l'applicazione funzioni correttamente nell'ambiente IDE, se essa è destinata a funzionare nei device mobile è necessario testarla anche in tale ambiente. Se è previsto l'uso sia da dispositivi iOS che Android, è necessario testarla su entrambi, in modo da verificare che le integrazioni con i plugin nativi funzionino correttamente in tutti i casi.

Il modo più semplice per testare l'applicazione nei propri dispositivi è quello di utilizzare InstaLauncher, come descritto nel libro: Launcher: funzioni native e pubblicazione negli store al paragrafo *Anteprima dell'applicazione in un launcher*. Se sono stati integrati plugin personalizzati, il debug andrà effettuato nel proprio Developer Launcher.

Quando InstaLauncher mostra l'anteprima dell'applicazione, esso è connesso all'IDE allo stesso modo dell'anteprima nel browser. Sarà quindi possibile effettuare il debug come illustrato nei paragrafi precedenti.

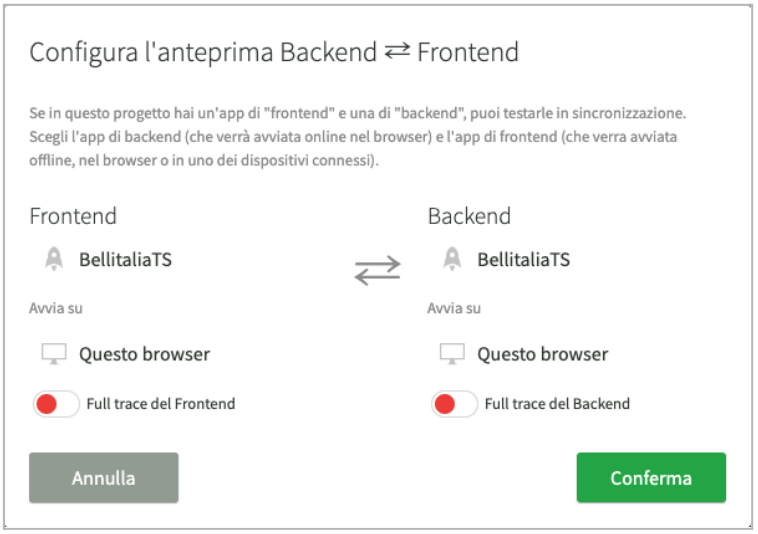
Debug di applicazioni locali o offline

Normalmente si tende a sviluppare l'applicazione utilizzando il modello online, perché è più immediato visto che non richiede l'utilizzo di particolari servizi o collegamenti a backend nel cloud.

Se la propria applicazione dovrà essere installata come locale in un dispositivo, sarà necessario verificarne il comportamento anche in questa condizione. A tale scopo è possibile attivare la modalità di anteprima *offline* sia nel browser che su InstaLauncher. In questa modalità l'applicazione è completamente contenuta nel dispositivo o nel browser e può accedere ai database locali SQLite.

Normalmente il funzionamento *offline* richiede un server nel cloud a cui accedere tramite API o sincronizzazione o infine Document Orientation remota, come spiegato nel libro relativo alla sincronizzazione. Se il backend è già stato sviluppato, testato ed installato è conveniente effettuare il debug direttamente nella modalità *offline*.
Se il backend non è ancora pronto, è disponibile un'ulteriore modalità di debug che consente di testare insieme in anteprima sia il backend che l'applicazione in modalità *offline*. In questo modo è possibile effettuare il debug di entrambe le configurazioni allo stesso tempo, potendo così correggere anche eventuali problemi di comunicazione fra le due parti.

Questo tipo di anteprima si chiama *Backend* <-> *Frontend*, ed è attivabile sempre dal menu del pulsante di avvio dell'anteprima. La prima volta che viene attivata, appare la seguente pagina di configurazione in cui è possibile scegliere quale applicazione deve essere avviata come *Frontend* e quale come *Backend*.



Normalmente entrambe le anteprime appaiono in due finestre browser affiancate. Se un InstaLauncher è collegato all'IDE, sarà possibile usarlo per visualizzare il *Frontend*.

In questa modalità la console di debug dell'IDE raccoglie informazioni da entrambe le applicazioni e le mostra affiancate.

Nota bene: la sessione di backend lanciata dall'anteprima *Backend* <-> *Frontend* è contemporaneamente una sessione browser e una sessione proxy. Questo implica che vengono lanciati sia l'evento *app.onStart* che l'evento *app.sync.onConnect.*

Se nell'evento *app.onStart* vengono inizializzate delle variabili globali, usate anche nei documenti o nel codice di sincronizzazione, l'applicazione potrebbe mostrare un comportamento diverso da quella che verrà installata, perché in tal caso le sessioni browser e quelle proxy sono completamente separate.

Si consiglia quindi di verificare che le variabili globali a livello di applicazione siano impostate sia nell'evento *app.onStart* che in *app.sync.onConnect*.

Reset dei database locali

Durante il debug di un'applicazione che utilizza database locali, può essere comodo cancellare il contenuto degli stessi in modo da testare nuovamente cosa succede la prima volta che l'applicazione viene lanciata. Per questo scopo è possibile utilizzare il metodo *App.<Database>.resetSchema* che cancella l'intero database locale. Questo metodo funziona solo se il database è locale al dispositivo. Se viene chiamato da un'applicazione online, viene solamente emesso un warning.

Debug di una server session

Il ciclo di vita di una sessione server è lo stesso di quelle browser; l'unica differenza è che la sessione server non ha un'interfaccia utente e viene mantenuta viva dal sistema.

Sia la sessione server che quelle browser iniziano con l'evento *app.onStart,* nel quale viene controllato il tipo di sessione tramite il metodo *app.isServerSession* che restituisce *true* in tale caso.

Se la sessione è di tipo server, solitamente si apre una videata diversa dal caso browser, che contiene i timer e i metodi da eseguire nel caso server. Tale videata non appare in nessun browser, tuttavia essa è perfettamente operativa.

Il debug della server session è quindi molto semplice: è sufficiente modificare il codice di *app.onStart* per eseguire forzatamente il codice relativo all'inizializzazione della server session, come mostrato nel seguente esempio:

```
App.Session.prototype.onStart = function (request)
{
  ...
  if (app.isServerSession() || true) {
    // inizializzo server session
  }
  else {
    // inizializzo browser session
  }
  ...
}
```

In questo modo, lanciando l'anteprima dell'applicazione, essa funzionerà in modalità *server session* e si potrà eseguire il debug con gli stessi strumenti visti fino ad ora, con l'ulteriore vantaggio che la videata della server session è effettivamente visibile e può essere utilizzata come ulteriore strumento di logging delle operazioni.

Debug di sessioni rest

Le sessioni rest entrano in gioco quando l'applicazione espone API, come abbiamo illustrato nel libro: Web API e File System. È possibile esporre due tipi di API: quelle gestite dall'evento *app.onCommand* e quelle di tipo OData.

L'API è un mezzo per lanciare metodi interni all'applicazione che poi eseguono i comandi richiesti e ne restituiscono i risultati.

Il debug di questo tipo di operazioni deve essere eseguito in due fasi: nella prima si testano i metodi interni che eseguono i comandi, e questo è possibile farlo tramite la normale anteprima dell'IDE, in cui si predispone una videata che richiama i comandi interni a partire da pulsanti o campi di input.

Nella seconda si testa il canale di comunicazione, quello che consente ai client esterni di consumare le API messe a disposizione. Il debug del canale di comunicazione può essere eseguito dall'anteprima se l'API viene gestita tramite *app.onCommand*, come spiegato nel paragrafo Testare la Web API esposta del libro sulla Web API.

Se la Web API è di tipo OData, invece, il canale di comunicazione può essere testato solo quando l'applicazione è installata su un server. Tuttavia, essendo questo un canale standard, non è necessario un vero e proprio debug in quanto il codice che gestisce l'API OData è già stato verificato. Si consiglia di leggere il paragrafo Testare la Web API OData esposta del libro suddetto.

Suggerimenti per l'ottimizzazione

In questo paragrafo viene elencata una serie di situazioni in cui è possibile incorrere durante le operazioni di debug. Per ognuna di queste verrà illustrata la strada migliore per affrontarle.

Lentezza nella visualizzazione dei dati

In alcune situazioni una datamap può risultare troppo lenta nel caricamento dei dati o nella visualizzazione, o l'interfaccia utente può diventare poco responsiva. In questi casi è necessario ottimizzare la datamap e, per questo, si consiglia la lettura del paragrafo Ottimizzazione delle performance del libro sulle datamap. Vediamo ora le possibili cause.

Troppo tempo prima che appaia il primo dato della lista

Se la videata richiede troppo tempo prima di far apparire i dati, è possibile che la query di estrazione carichi troppi dati. Per risolvere è possibile:

- Inserire opportuni filtri.
- Utilizzare la proprietà *maxRows* della datamap.
- Utilizzare il caricamento dati a blocchi, tramite *dataPageSize*.

Per verificare il caso, si può leggere la query eseguita tramite la console di debug e poi eseguirla nel query editor dell'IDE per vedere quali risultati estrae. Nell'evento *afterLoad* della datamap è possibile anche scrivere nel log la proprietà *this.length* per vedere quanti record sono stati estratti.

È sempre consigliabile estrarre il numero minimo di dati possibile e una buona regola è che siano meno di 1000. Se è necessario estrarre più dati, considerare il caricamento a blocchi.

Interfaccia utente poco responsiva

Questo può succedere se vengono estratti tanti dati, ad esempio 1000, e non è stata attivata la paginazione incrementale o quella con window. In questo caso la datamap carica tutte le righe nel DOM del browser che può diventare lento o poco responsivo.

Per verificare il caso è sufficiente scrivere nel log il numero di dati estratti. Se è significativamente superiore a 100 è necessario attivare la paginazione impostando la proprietà *pageSize* della datamap, ad esempio al valore 50, o attivando la proprietà *useWindow*. Si ricorda che in questi casi potrebbe essere necessario modificare la struttura della pagina per far sì che la paginazione funzioni correttamente.

Lentezza nella visualizzazione dei dati

Se i dati compaiono lentamente e non si ricade nei casi precedenti, è necessario verificare l'evento *onRowComposition* della datamap. Questo evento viene notificato prima di inviare ogni riga al browser, quindi se nel codice di gestione dell'evento vengono eseguite delle query o dei caricamenti o altre operazioni asincrone non immediate, i dati possono apparire lentamente.

Per verificare il caso, basta osservare la console di debug mentre la datamap viene caricata. Se appaiono tante query, significa che il codice di visualizzazione le richiede. Per risolvere il problema è necessario pre-caricare i dati da visualizzare in modo da non doverlo fare durante l'evento *onRowComposition*. Nel caso di datamap basate su query, i dati aggiuntivi possono essere caricati inserendo opportune join nella query; se invece la datamap è basata su documenti, è possibile utilizzare proprietà derivate o inserire una query di caricamento del documento che valorizza opportune proprietà unbound che potranno poi essere usate durante la visualizzazione.

Lentezza particolare su dispositivi Android

Se la videata funziona bene nel browser e nei dispositivi iOS, ma risulta lenta su Android, la causa può risiedere in una particolare complessità della riga template che compone la datamap.

La causa principale di questo problema risiede nel fatto che la webview di sistema dei dispositivi Android non utilizza in modo particolarmente spinto le risorse hardware del device. Per testare questa affermazione è sufficiente eseguire un test di velocità del

JavaScript usando Chrome nel dispositivo per confrontarlo con i risultati ottenuti su iOS o nel browser del desktop. Anche l'ormai obsoleto sunspider può dare risultati interessanti, solitamente compresi tra un quarto e un decimo.

Le applicazioni ibride su dispositivi Android devono quindi essere particolarmente ottimizzate, e se le soluzioni consigliate ai paragrafi precedenti non sono stati sufficienti, si consiglia di semplificare al massimo la composizione del template della datamap, fino a portarlo ad un unico oggetto, il cui contenuto potrà essere definito in modo complesso nell'evento *onRowComposition* e poi assegnato alla proprietà *innerHTML* del template. In questo modo nel browser si ha un unico widget nel DOM virtuale il cui contenuto complesso viene espresso direttamente in HTML.

Se si utilizza questa tecnica, ci si deve ricordare di proteggere i dati dell'utente da attacchi *Xss Injection*, validando ogni dato non costante tramite la funzione *App.Utils.htmlEscape* prima di inserirlo nella stringa HTML da assegnare al template.

In questo caso, inoltre, sarà possibile intercettare i clic sugli elementi interni al template, inserendo opportuni *id* per ogni oggetto nel codice HTML del template e poi verificando la proprietà *event.target* dell'evento *onClick* definito sul template.

Modifiche ad un grande numero di record

In alcune situazioni, l'aggiornamento, inteso come inserimento modifica o cancellazione, di un grande numero di record (>100) nello stesso script può richiedere più tempo del previsto, sia usando query che documenti.

Innanzitutto si deve osservare che se non è richiesta la sincronizzazione dei database locali, la modifica dei dati può avvenire anche tramite query e non solo tramite documenti. In questo caso, il modo più veloce è quello di modificare i dati con una query unica; questo è possibile in molti casi usando istruzioni di update con subquery al posto di valori costanti.

Se l'utilizzo di una query unica non è possibile, l'alternativa è quella di eseguire un ciclo di istruzioni di aggiornamento dati o di salvataggio di documenti. In questo caso si potrebbero verificare tempi più lunghi del previsto. Per ottimizzare i cicli di modifica dei dati è possibile utilizzare una o più delle seguenti tecniche.

Accesso con chiave primaria o con indice

Occorre essere sicuri che tutti i comandi di modifica o cancellazione dati contengano un filtro sulla chiave primaria o sulla chiave di un altro indice definito nella tabella. Senza indice, infatti, il database deve effettuare la scansione dell'intera tabella per sapere quali dati deve aggiornare.

Transazione unica

Se le modifiche ai dati **non** avvengono tutte all'interno della medesima transazione, il database è costretto a eseguire una transazione per ogni modifica. In particolare, la fase di conferma della transazione (commit) richiede una scrittura sincronizzata del log delle transazioni, e questo è il più lento fra i tipi di accesso al file system.

Si consiglia quindi di aprire esplicitamente la transazione sul database prima di iniziare il ciclo di modifiche, sia via query che tramite documenti, e di confermare esplicitamente la transazione al termine. Per maggiori informazioni si consiglia di leggere i paragrafi Transazioni e Gestione delle eccezioni del libro *Struttura del database*.

Numero di query elevato

Nel caso di applicazioni che dovranno essere installate su un server cloud e che verranno accedute da decine o centinaia di utenti contemporanei, occorre una particolare attenzione all'ottimizzazione dell'accesso ai dati.

Infatti l'architettura modulare su cui si basa il framework di Instant Developer Cloud rende possibile gestire tutte queste sessioni però, se esse accedono troppo frequentemente al database, si può creare un collo di bottiglia proprio nell'accesso ai dati.

È possibile potenziare il server, e quindi il database server, ma nel cloud le risorse hanno un costo proporzionale alla quantità utilizzata, quindi è conveniente una fase di ottimizzazione dell'applicazione, che pertanto deve partire proprio dalla fase di accesso ai dati.

Ci sono due segnali di attenzione che richiedono un'azione immediata, fin dalla fase di sviluppo, prima ancora dei test di carico. Essi sono:

- Durante l'uso dell'applicazione in anteprima dall'IDE, il log della console si riempie di query.
- Durante l'uso dell'applicazione in anteprima dall'IDE, alcune operazioni richiedono tempi dell'ordine di centinaia di millisecondi e il tempo di attesa è legato proprio all'accesso ai dati.

Questi sono segnali di attenzione perché, quando l'applicazione sarà in produzione, il carico di una sessione verrà moltiplicato per il numero di utenti collegati, portando il sistema molto velocemente alla saturazione.

Le tecniche per migliorare l'accesso ai dati sono le seguenti:

- Ottimizzare le query perché utilizzino sempre indici.
- Ridurre il numero di query di caricamento dati correlati usando proprietà derivate o query di caricamento dei documenti, come già visto nei casi precedenti.
- Utilizzare cache a livello di sessione: piuttosto che caricare sempre i dati dal database, è bene mantenerli nella memoria della sessione in modo da averli sempre disponibili. Il compromesso fra uso del database e uso della memoria è spesso a favore della seconda.
- Utilizzo di cache a livello di applicazione (App): i metodi *getRelated* e *loadByKey* dei documenti possono utilizzare una cache che evita di dover ricaricare i dati acceduti più frequentemente. È possibile implementare anche una propria cache di dati a livello di applicazione (App) che verrà condivisa fra tutte le sessioni di un processo worker.

Se nessuna di queste tecniche è sufficiente, si consideri di passare ad un'architettura con database locali sincronizzati, in cui ogni sessione è completamente autonoma e non carica il server nel cloud.

Dati mancanti in database locali sincronizzati

Questo tipo di problematica si può verificare in un'applicazione locale, installata nei dispositivi mobili, che utilizza il sistema di sincronizzazione per mantenere allineati i database locali con il cloud.

Quando si verifica in ambiente IDE o è riproducibile, il problema è facilmente affrontabile con le tecniche viste in precedenza. Se invece avviene in produzione, in modo apparentemente casuale e con bassa frequenza, l'identificazione della causa diventa più sfuggente. In questo paragrafo vedremo come identificare le possibili cause anche in queste condizioni.

Preparazione dell'ambiente di tracciamento

La prima fase per l'identificazione di queste problematiche consiste nell'implementare i seguenti sistemi di tracciamento:

1) Le sessioni proxy devono avere il log strutturato attivo e il nome della sessione deve includere il nome dell'utente (o un suo codice identificativo).
2) Le applicazioni mobile devono avere attivo il servizio Analytics.
3) Il corretto funzionamento della fase di resincronizzazione dei database locali (*onResyncClient*) deve essere certificato.
4) Nell'applicazione locale deve essere implementato l'evento *app.sync.DO.onError*, in cui si deve usare il metodo *app.sync.Log.error* per informare la sessione proxy dell'errore avvenuto nel dispositivo. Si consiglia di implementare anche l'evento *app.sync.DO.onSync* controllando *option.status* per loggare un eventuale errore nella sincronizzazione iniziale.
5) Nella sessione proxy deve essere implementato l'evento *app.sync.DO.onVariationFiltering* in cui deve essere loggata la variazione nel caso in cui il parametro *options.match* sia *true*.
6) Nel sistema deve essere temporaneamente disabilitata la compressione delle differenze tramite *minimizeVariations.*
7) Nell'applicazione mobile deve essere presente un pulsante o una voce di menu che forza la resincronizzazione completa del database locale chiamando il metodo *resyncAllClasses* passando come opzioni *compareData:true e dataDiff:diffcoll*, dove *diffcoll* è una collection vuota definita prima di chiamare l'evento. La collection *diffcoll* viene riempita dal metodo *resyncAllClasses* con l'elenco delle differenze fra il database locale prima e dopo la resincronizzazione. Tale collection dovrà essere inviata al server tramite *app.sync.Log.warn*.

Si consiglia di leggere il paragrafo: Analisi della sincronizzazione a runtime del libro *Sincronizzazione.*

Identificazione dei problemi relativi alla sincronizzazione

Il sistema di sincronizzazione si basa sul corretto funzionamento della sessione proxy e dei documenti dell'applicazione.

Il primo passo per identificare i problemi è quindi quello di eliminare qualunque eccezione o warning contenuto nei log strutturati delle sessioni proxy, oltre a tutte le eccezioni segnalate dal sistema Analytics, almeno quelle che riguardano i documenti dell'applicazione e quelle relative alla sincronizzazione, riportate nella pagina Analytics relativa alla sincronizzazione.

Si può essere quindi pronti per l'identificazione di problemi relativi a dati mancanti quando, oltre ai sette punti visti in precedenza, si verificano le seguenti condizioni:

1) Non ci sono eccezioni e warning nel log strutturati delle sessioni proxy.
2) Non ci sono eccezioni segnalate da Analytics nei dispositivi mobile (almeno quelle che riguardano i documenti).
3) I database locali sono sincronizzati correttamente. Questo si può ottenere cancellando i dati dell'applicazione o usando il pulsante o la voce di menu inserita per forzare la resincronizzazione.

A questo punto, nel momento in cui viene segnalata una situazione di dati mancanti, si deve procedere come segue:

1) Utilizzare sul dispositivo in cui mancano i dati il pulsante o la voce di menu per forzare la resincronizzazione.
2) Identificare nel log strutturato l'ultima sessione proxy dell'utente e scaricare la lista delle differenze fra il contenuto del database locale precedente e successivo alla resincronizzazione.
3) Identificare se nell'elenco compaiono i dati mancanti segnalati. Se non compaiono significa che i dati sono nel dispositivo, ma per qualche ragione l'applicazione non li mostra. In questo caso il problema è da ricercare nel codice dell'applicazione.
4) Se il dato mancante è nella lista, occorre verificare che nella tabella *z_syncdo* del server siano contenuti i record relativi alla sincronizzazione differenziale di questo documento. Si deve quindi effettuare una query come la seguente:

```
select
  *
from
  z_syncdo
where body like '%n32F57BgnaEzyO+sbfAuPw==%'
```

   In cui **`32F57BgnaEzyO+sbfAuPw==`** deve essere sostituito con la chiave primaria GUID del documento mancante.
5) Se le variazioni non sono presenti o non sono coerenti con i dati che quel documento dovrebbe avere, ci possono essere due casi:
   a) I dati mancano su tutti (o quasi) i dispositivi: in questo caso l'errore è nella sessione che ha generato i dati. Occorre verificare se ci sono state eccezioni nei log di tale sessione, o se tali eccezioni siano state mascherate dal codice dell'applicazione.
   b) I dati mancano solo sul dispositivo segnalato. In questo caso è possibile che i topics del dispositivo non siano coerenti con le differenze registrate.
6) Per verificare se il problema sono i topics, utilizzando il log strutturato occorre selezionare tutte le sessioni proxy del dispositivo con i dati mancanti. Per ogni sessione proxy, controllare il log dell'evento *onVariationFiltering* per vedere se le variazioni contenute nella tabella *z_syncdo* per il documento mancante sono state inviate al dispositivo. In caso negativo, il problema è da ricercare nell'impostazione

dei topics della sessione di sincronizzazione. In caso positivo, occorre verificare perché la variazione ricevuta non è attualmente presente nel terminale.

7) Se una variazione ricevuta non è presente nel database locale del terminale, occorre ricercare due possibili cause:
    a) È avvenuta un'eccezione durante il salvataggio dei dati, e questo deve risultare nel sistema Analytics.
    b) Il codice dell'applicazione manipola i dati del database senza direttamente utilizzare i documenti, ad esempio tramite comandi di aggiornamento dati.

Risoluzione dei problemi relativi alla sincronizzazione

Da un'analisi statistica dei problemi riportati al servizio di assistenza relativi alla sincronizzazione, le cause principali di dati mancanti sono le seguenti:

1) 60% eccezioni non gestite nella sessione proxy o nella sessione locale.
2) 30% errore nella definizione dei topics dei documenti o della sessione.
3) 10% errore di salvataggio dei documenti nel dispositivo.

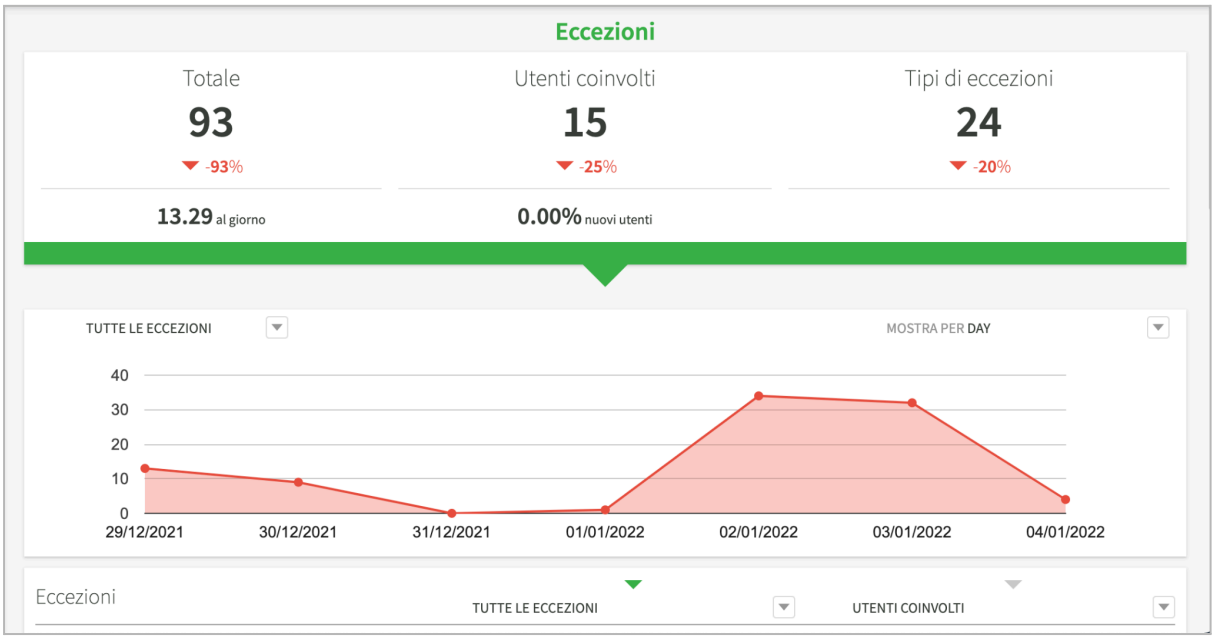
Le prime due cause possono essere risolte in autonomia dal programmatore. La terza causa, identificata da eccezioni di tipo *Database not open* o similari nel sistema Analytics o nei log relativi agli errori della sincronizzazione può richiedere una consulenza specifica del servizio di supporto di Pro Gamma. Prima di aprire un ticket di richiesta di supporto si consiglia quindi di seguire tutte le procedure descritte in questo paragrafo anche per velocizzare i tempi di risoluzione del problema.

Debug in ambiente di produzione

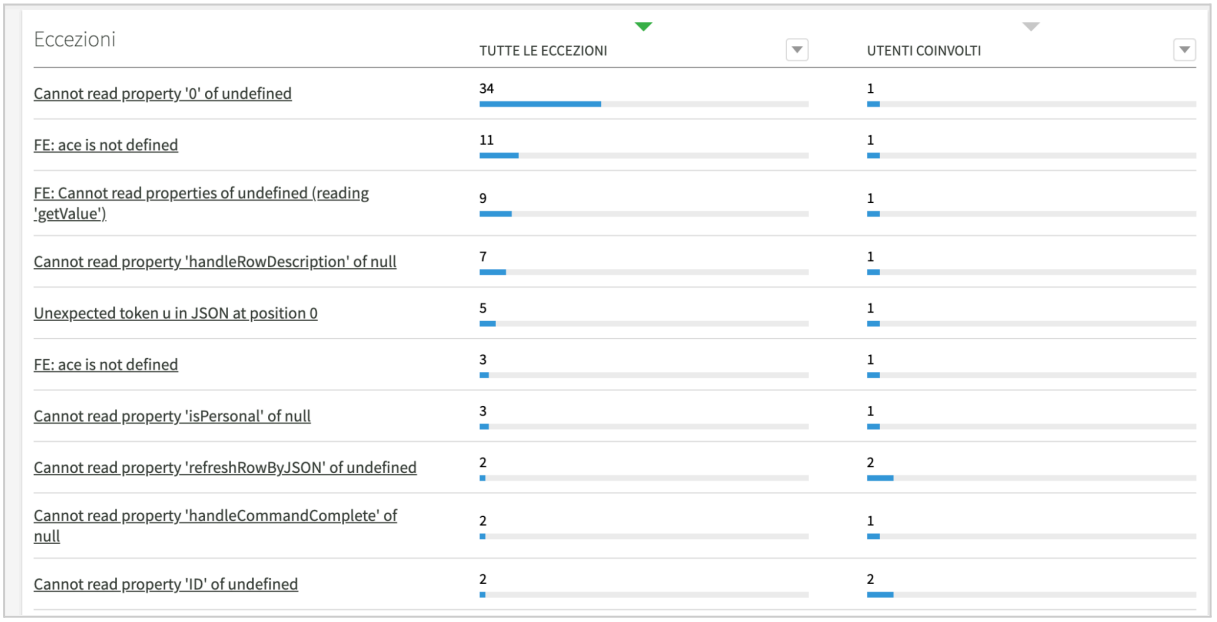
Quando l'applicazione entra in produzione e gli utenti finali cominciano ad utilizzarla, si verificano problemi o errori non previsti. In questa fase è necessario avere una strategia di debug per scoprire e risolvere più velocemente possibile queste situazioni.

La prima fase consiste nel rilevare gli errori che avvengono in produzione. Per le applicazioni installate nel cloud, il metodo migliore è quello di attivare il log strutturato come indicato nel paragrafo corrispondente del libro I server di produzione. Tramite il log strutturato sarà possibile vedere sessione per sessione quali errori o warning vengono emessi.

Per le applicazioni mobile, ma anche per quelle installate nel cloud, uno strumento di rilevamento degli errori è il servizio Analytics che rileva le eccezioni JavaScript che avvengono nelle applicazioni, anche quando sono offline. L'immagine seguente mostra un esempio di pagina relativa alla gestione delle eccezioni di Analytics.

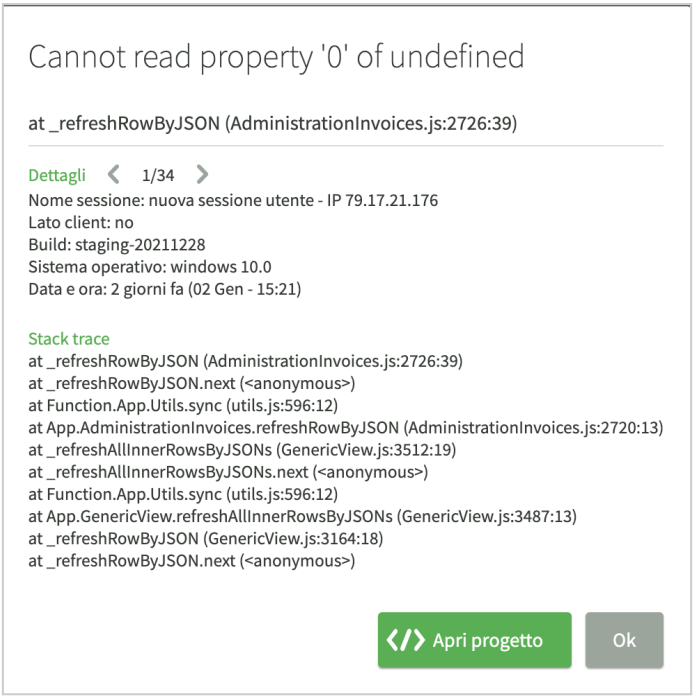


La lista delle eccezioni sotto al grafico permette di conoscere il numero di eccezioni avvenute per tipo e quanti utenti diversi hanno incontrato il problema.



Per ogni eccezione, infine vengono mostrate informazioni specifiche, come esemplificato nell'immagine alla pagina seguente. Nel riquadro risultano particolarmente importanti il nome del metodo e il nome del file JavaScript in cui è avvenuta l'eccezione. Dal nome del file si può risalire alla classe che contiene il metodo e a quel punto è possibile sapere dove investigare il problema.

Esiste anche un metodo particolare per aprire il progetto direttamente alla riga di codice che ha generato l'eccezione, che vedremo nel paragrafo successivo.



Debug delle sessioni in produzione

Durante l'installazione dell'applicazione è possibile attivare il flag *Abilita il debug a runtime*, che aggiunge al codice compilato per l'applicazione anche il supporto per il debug a runtime.

Tale supporto non penalizza di per sé le prestazioni dell'applicazione perché, pur essendo presente, non è normalmente attivo. Si deve tenere presente tuttavia che abilitando il debug a runtime, verrà effettuata una copia di backup del progetto nello stato in cui è stata iniziata l'installazione per poi poterla aprire e collegarla alla sessione installata sul server. Tale copia di backup rimane nel proprio server IDE e può occupare una determinata quota di spazio.

Per le applicazioni installate in un server cloud, è possibile iniziare una sessione di debug a runtime dalla videata che presenta la lista delle sessioni del log strutturato. Se una sessione è online, cioè è ancora attiva, e l'applicazione ha il supporto per il debug, l'icona del pulsante verde sulla riga della sessione diventa quella del debugger. Cliccando sul pulsante si aprirà una sessione IDE in sola lettura con la copia di backup progetto, inoltre verrà attivato il supporto di debug per la sola sessione selezionata.

Nell'IDE sarà quindi possibile vedere la console e il flusso del codice in corrispondenza alle azioni dell'utente nonché utilizzare il query editor dell'IDE questa volta collegato al database di produzione. Chiudendo la sessione IDE, il supporto di debug verrà disattivato e la sessione utente ritornerà allo stato normale. Durante il debug a runtime, la sessione in fase di debug avrà prestazioni minori.

Debug a runtime per applicazioni locali

È possibile attivare il supporto per il debug anche in caso di applicazioni compilate per launcher. In questo caso però non sarà possibile entrare direttamente nel debug della

sessione del device, ma sarà possibile attivare un parametro dell'applicazione per raccogliere più precisamente la riga di codice che ha generato l'eccezione in Analytics.

Impostando il parametro di applicazione *trackCodeLine* a *true*, nella videata di dettaglio dell'eccezione vista prima sarà presente un ulteriore pulsante che permetterà di aprire la copia di backup del progetto relativa all'applicazione installata, già posizionata sulla riga di codice che ha generato eccezione o comunque più vicino possibile ad essa se l'eccezione è avvenuta internamente o durante un'operazione asincrona.

Test automatico delle applicazioni

Immaginiamo che, dopo aver preparato la prima versione dell'applicazione ed aver eseguito con successo le fasi di alfa e beta test, l'applicazione venga installata in produzione e gli utenti finali inizino ad utilizzarla.

Anche quando la prima versione è completa, il lavoro di sviluppo e quindi di modifica del codice non è terminato. Infatti:

- La correzione degli errori segnalati dai sistemi di rilevazione (Analytics e log strutturato) richiede modifiche del codice.
- Il feedback degli utenti porta ad implementare modifiche o nuove funzionalità.
- La roadmap del progetto prevede versioni successive.
- L'evoluzione dei sistemi operativi cloud e mobile richiede aggiornamenti al framework di base e quindi al codice dell'applicazione.

Ogni modifica al codice, prima di essere installata in produzione, richiede un test parziale o completo perché non è facile tenere conto di tutte le specifiche anche durante la modifica di una singola riga di codice. D'altra parte un test completo può risultare molto oneroso e non proporzionato all'entità della modifica che si sta per portare in produzione.

Il sistema di test automatico di Instant Developer Cloud nasce proprio per minimizzare i rischi legati alle modifiche al codice dell'applicazione senza dover perdere tempo e risorse preziose.

Oltre alla possibilità di implementare test automatici di *non regressione*, volti ad assicurare il più possibile il comportamento corretto dell'applicazione, il sistema di test di Instant Developer Cloud risolve un ulteriore problema di fondamentale importanza nell'ambito dei servizi cloud: la stima e l'ottimizzazione delle risorse necessarie al funzionamento.

Infatti nel momento in cui l'applicazione è pronta, si deve procedere all'acquisto delle risorse cloud per la gestione del servizio, ed occorre stimarne la quantità in funzione del numero atteso di utenti contemporanei. La supposizione che il cloud presenta *elasticità* e quindi si adatta in automatico al carico in entrata, da un lato è vera, ma se viene applicata ad un'applicazione non ottimizzata può causare un'impennata imprevista nei costi del cloud, e risolvere questi problemi a posteriori è sempre più difficile ed urgente.

Per questa ragione il sistema di test di Instant Developer Cloud è in grado di simulare un qualunque carico di utenti su un determinato server di test, permettendo così di

comprendere quali parti dell'applicazione dovranno essere ottimizzate e quante risorse saranno necessarie per gestire i carichi attesi. Oltre alla modalità di *non regressione*, è quindi disponibile il *test del carico*.
Data l'importanza di queste problematiche, il sistema di test automatico di Instant Developer Cloud è incluso nella versione base delle licenze di sviluppo, senza richiedere costi aggiuntivi.

Nota bene: installare in produzione un'applicazione senza aver implementato gli opportuni test di non regressione e di carico è una pratica altamente sconsigliata che scarica i problemi sugli utenti finali e li rende più urgenti. Per questa ragione, il supporto tecnico di Pro Gamma non riconosce automaticamente l'urgenza delle richieste di segnalazione di problemi sulle proprie applicazioni se non è stato implementato un opportuno insieme di test di non regressione o di carico.

Utilizzo del sistema di test automatico

Il sistema di test automatico di Instant Developer Cloud si basa su tre fasi:

1) Registrazione di sessioni campione ed identificazione dei risultati attesi.
2) Organizzazione delle sessioni in suite di test.
3) Esecuzione automatica delle suite di test in modalità non regressione o di carico.

La registrazione delle sessioni campione avviene usando l'applicazione stessa: vengono simulati gli stessi processi dell'utente e quindi può essere testato globalmente il funzionamento dei componenti applicativi già in forma integrata. Se si desidera implementare test unitari, è possibile realizzare apposite videate che effettuano i test di base. Tali videate saranno disponibili solo nell'ambiente di test e non in produzione.

Per le applicazioni basate su dati, uno dei problemi fondamentali del test sono i dati di partenza, che devono essere sempre coerenti per poter valutare correttamente l'output del test automatico.

Per questa ragione, il sistema di test di Instant Developer Cloud permette di identificare un particolare backup del database come insieme di dati di base da cui iniziare i test. Prima di lanciare una determinata suite di test, il database dell'applicazione potrà essere automaticamente ripristinato in base alle impostazioni fornite.

Vediamo ora nel dettaglio come si svolgono le varie fasi di registrazione e riproduzione dei test di un'applicazione.

Registrazione di una sessione di test

La registrazione di una sessione di test inizia dal sottomenu *test automatici / sessioni* del menu di progetto. Per poter effettuare una registrazione è necessario aver installato l'applicazione su un server di produzione nel cloud di Instant Developer e che tale server sia stato marcato come *server utilizzabile per il test* nelle impostazioni del server.

È infatti importante tenere separati gli ambienti di test da quelli di produzione. Il sistema di test automatico nasce per testare le nuove versioni prima di rilasciarle agli utenti; inoltre le opzioni di ripristino di un backup del database non consentono di utilizzarlo sui server di produzione veri e propri.
Occorre tenere conto, infine, che un test di carico deve avvenire in condizioni standard e che esso carica il server fino alla saturazione e anche oltre, quindi durante il test di carico tale server non può essere utilizzato dagli utenti dell'applicazione.

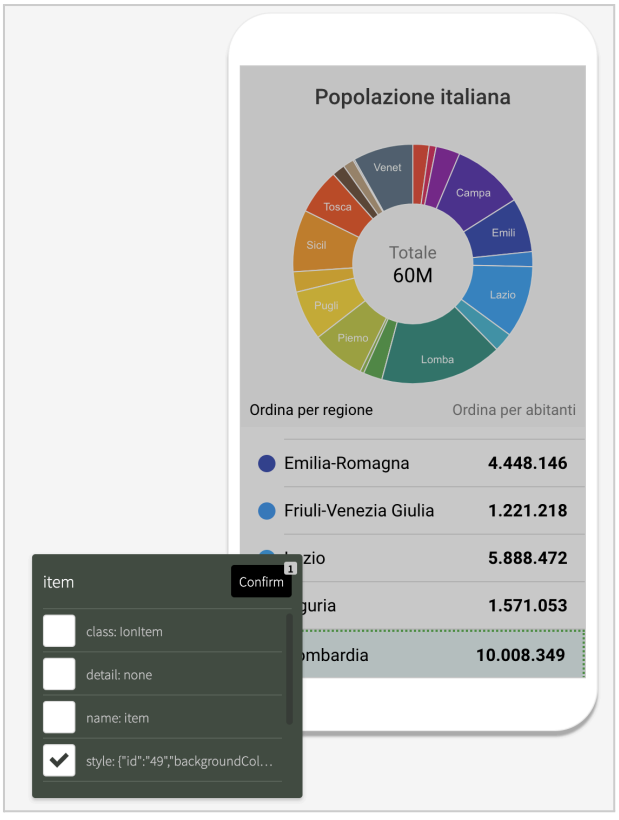
Dopo aver indicato un titolo per la sessione di test, aver scelto l'applicazione da testare e aver selezionato l'installazione da utilizzare, è possibile cliccare il pulsante *Registra* per iniziare la registrazione.

Durante la registrazione sarà possibile utilizzare l'applicazione in modo normale, ma la barra laterale destra del browser conterrà i comandi per la gestione della sessione di test.
Mettendo in pausa la registrazione, infatti, si potranno identificare eventuali risultati corretti in funzione delle azioni dell'utente.

Al termine della registrazione, cliccare il pulsante *stop* per concludere la registrazione e registrare i risultati. È possibile verificare la registrazione di una sessione di test tramite la voce *replay* del menu di riga della sessione di test.

Identificazione dei risultati attesi

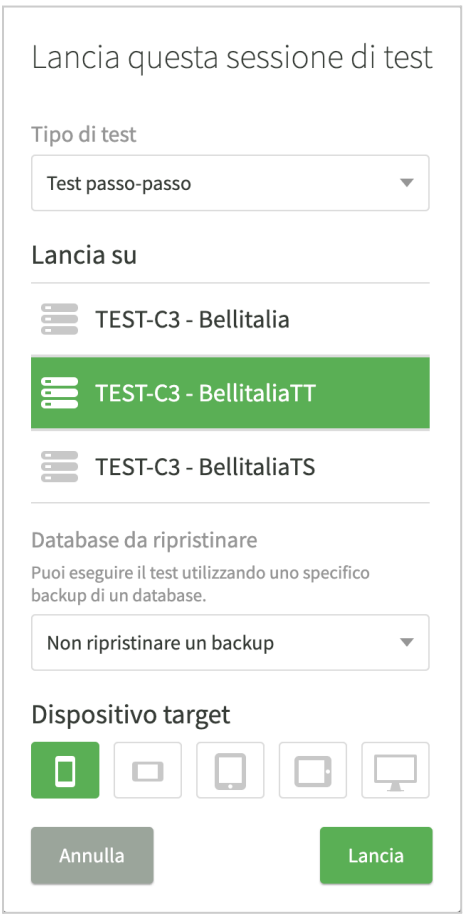
Mentre la registrazione della sessione campione è in pausa, è possibile cliccare su un oggetto visuale e definire quali proprietà dovranno essere verificate durante il test automatico. Vediamo un esempio:



Nell'immagine precedente la registrazione è stata messa in pausa dopo aver cliccato sulla fetta del grafico a torta relativa alla regione Lombardia. Si vuole verificare che la lista venga posizionata esattamente su tale regione e che lo sfondo della riga cambi colore. Per identificare questo risultato è sufficiente cliccare sullo sfondo della riga e poi selezionare nella lista delle proprietà lo *style*, che, appunto, identifica l'apposizione del colore di background corretto. Cliccando il pulsante *Confirm* il risultato da controllare verrà effettivamente acquisito. Può essere interessante identificare anche il totale mostrato nel centro del grafico. In tal caso la proprietà da controllare sarà *innerHTML*. Tutti gli oggetti per i quali è stato definito un risultato atteso verranno bordati in verde quando la registrazione è in pausa. È possibile poi riprendere la registrazione ed eseguire ulteriori azioni dell'utente e identificare altri risultati da controllare.

Esecuzione del test relativo ad una singola sessione

Cliccando sul pulsante *Esegui test* relativo ad una sessione registrata, sarà possibile rieseguire il test della singola sessione. Apparirà la seguente videata per la definizione dei parametri del test:



Il tipo di test prevede le opzioni *non regressione* e *passo-passo.* Nel primo caso la sessione verrà eseguita senza interfaccia utente e si potranno leggere i risultati al termine del test. Nel secondo caso, invece, la sessione verrà eseguita aprendo un browser e visualizzando i passaggi della sessione e le verifiche effettuate. In questo caso sarà possibile selezionare anche un dispositivo target. I *test di carico* sono disponibili solo all'interno di una suite di test.

Definizione di una suite di test

Una suite di test è un insieme di sessioni che vengono riprodotte per testare compiutamente l'applicazione o una parte di essa. La creazione di una suite avviene dalla pagina delle suite di test che appare cliccando sulla voce *Test automatici* del menu di progetto.

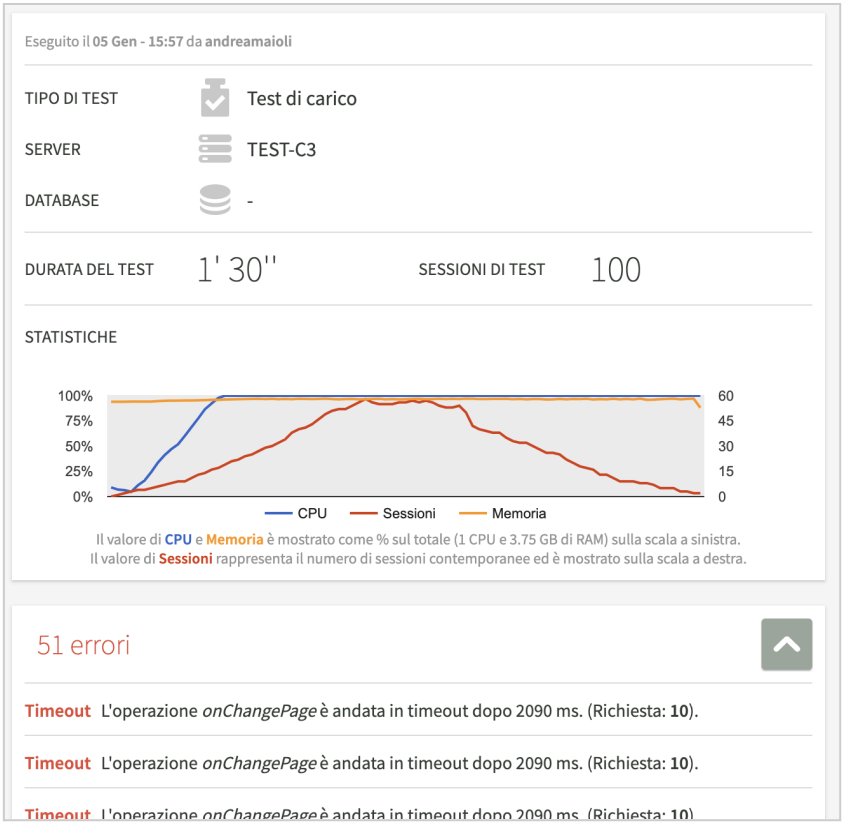
Definire una suite di test richiede i seguenti passaggi:

- Inserire il titolo, scegliere il tipo di test e l'applicazione da testare. I tipi disponibili sono: *test di carico* o *test di non regressione*.
- Scegliere l'installazione di default su cui eseguire la suite di test.
- Scegliere quale backup di dati ripristinare prima di eseguire la suite.
- Nel caso di test di carico, inserire il numero di sessioni da simulare e la durata.
- Aggiungere una o più sessioni di test da eseguire.

Tutti i dati della suite, l'elenco e l'ordine delle sessioni sono modificabili direttamente dalla pagina della lista delle suite, anche dopo averle definite.

Esecuzione di una suite di test

L'esecuzione di una suite avviene cliccando il pulsante *Esegui test* sulla riga della suite desiderata. Verrà chiesta conferma dell'installazione su cui eseguire il test, poi verrà mostrata la pagina dei risultati in cui apparirà la riga relativa al test in corso. Al termine del test la videata della console mostrerà un toast di notifica e sarà possibile vedere il dettaglio dei risultati.



Per i test di carico, i risultati riportano il grafico di utilizzo della CPU e della memoria in funzione del numero di sessioni utente contemporanee. Sotto al grafico è presente la lista degli errori avvenuti. In caso di test di carico, non verranno effettuati i controlli identificati durante la registrazione, ma verrà verificato che l'esecuzione abbia avuto luogo correttamente, che non ci siano state eccezioni e che le operazioni non abbiano richiesto un tempo significativamente più lungo di quello registrato. In questo caso, infatti, si avrebbe una esperienza utente degradata.

Nell'immagine alla pagina precedente viene mostrato un esempio di risultato. È facile vedere che la riga blu, che rappresenta il carico CPU, cresce molto velocemente rispetto al numero di sessioni, rappresentato dalla riga rossa. Questo significa che il server utilizzato ha troppe poche risorse per permettere al numero di utenti configurato nel test di eseguire l'applicazione contemporaneamente.

Nell'elenco degli errori sotto al grafico è possibile vedere quali azioni sono andate in timeout, cioè hanno richiesto molto più tempo del previsto. In un caso come questo, l'applicazione dovrà essere decisamente ottimizzata, oppure si dovrà ricorrere ad un server con maggiore capacità.

Per quanto riguarda i test di non regressione, la sezione del grafico non sarà presente, ma sarà possibile vedere gli eventuali errori dovuti ai controlli dei risultati identificati durante la registrazione.