Analisi e valutazione di soluzioni di reportistica centralizzata

Sviluppo di un Proof of Concept e confronto prestazionale tra JasperServer e soluzioni alternative

versione 0.7

Registro delle modifiche

Versione	Data	Descrizione
v0.7	2026-01-15	Concluso capitolo 5 e revisione documento
v0.6	2026-01-14	Concluso capitolo 6, sviluppo capitolo 5
v0.5	2026-01-13	Sviluppo capitolo 6
v0.4	2026-01-12	Conclusione capitolo 4
v0.3	2026-01-09	Conclusione capitolo 2, sviluppo capitoli 3-8, iniziato sviluppo capitolo 4
v0.2	2026-01-08	Sviluppo capitolo 2
v0.1	2026-01-07	Prima Bozza

Indice

1. Introduzione

1.1. Scopo del documento

Il tirocinio formativo è incentrato sulla necessità di analizzare soluzioni alternative a JasperServer a seguito del cambiamento delle policy di licensing, che ha comportato la non disponibilità della versione community. Tale cambiamento ha reso necessario individuare una piattaforma di reportistica centralizzata in grado di garantire funzionalità adeguate, sostenibilità economica e buone prestazioni, mantenendo al contempo un'elevata integrazione con i sistemi aziendali esistenti.

La reportistica rappresenta un elemento strategico per il supporto alle attività di analisi e ai processi decisionali aziendali. Per questo motivo, è stato avviato uno studio comparativo volto a valutare diverse soluzioni di reporting, analizzandone le caratteristiche tecniche, funzionali e prestazionali in un contesto di utilizzo comparabile a quello reale.

1.2. Obiettivi del PoC

L'obiettivo del Proof of Concept (PoC) è valutare in modo strutturato e oggettivo alcune soluzioni di reportistica centralizzata come possibili alternative a JasperServer, a seguito del cambiamento delle sue condizioni di utilizzo. In particolare, il PoC mira a verificare la fattibilità tecnica delle soluzioni analizzate, valutandone le modalità di installazione e configurazione, le funzionalità principali offerte e le possibilità di integrazione con le sorgenti dati e i sistemi aziendali.

Un ulteriore obiettivo del PoC è confrontare le prestazioni delle diverse piattaforme in scenari di utilizzo significativi, misurando aspetti quali i tempi di generazione dei report e il comportamento del sistema sotto carico. I risultati ottenuti consentono di evidenziare punti di forza, criticità e limiti di ciascuna soluzione, fornendo all'azienda elementi concreti a supporto del processo decisionale per l'eventuale adozione di un nuovo sistema di reportistica centralizzata.

2. Attività svolte

Sono state analizzate diverse soluzioni di reportistica alternative a JasperServer, valutandone le funzionalità principali, le modalità di integrazione e le prestazioni in un contesto di utilizzo simile a quello aziendale.

2.1. Analisi dei requisiti

La fase iniziale del lavoro ha previsto uno studio approfondito della piattaforma JasperServer, al fine di comprenderne le funzionalità principali, le modalità di utilizzo e i limiti emersi a seguito del cambiamento delle policy di licensing. Questa analisi ha consentito di individuare le caratteristiche ritenute fondamentali per una piattaforma di reportistica centralizzata in grado di soddisfare le esigenze aziendali.

Successivamente, attraverso un confronto con il tutor aziendale, sono stati definiti i requisiti funzionali e tecnici che le soluzioni alternative avrebbero dovuto soddisfare per essere considerate valide sostitute di JasperServer. Tali requisiti sono stati individuati tenendo conto dei flussi di lavoro esistenti, delle modalità di creazione dei report e delle esigenze di integrazione con i sistemi aziendali.

In particolare, è stata ritenuta necessaria la disponibilità di un editor grafico per la progettazione dei report, al fine di consentire la creazione e la modifica dei layout in modo intuitivo e coerente con quanto offerto da JasperServer. È stato inoltre considerato fondamentale il supporto all'input dei dati in formato JSON, in quanto largamente utilizzato per lo scambio di informazioni tra i servizi applicativi.

Un ulteriore requisito riguarda la possibilità di creare e gestire sottoreport all'interno di un report principale, funzionalità essenziale per la rappresentazione strutturata di dati complessi. È stata inoltre richiesta la possibilità di effettuare il deploy e la gestione della piattaforma su un server locale, evitando una dipendenza esclusiva da soluzioni cloud e garantendo un maggiore controllo sull'infrastruttura e sui dati.

Dal punto di vista dell'output, è stata considerata indispensabile la possibilità di esportare i report in molteplici formati, quali ad esempio PDF, Excel e altri formati comuni, al fine di supportare differenti esigenze di fruizione. È stato inoltre definito come requisito il supporto a chiamate API verso il server di reportistica, per consentire la generazione dei report in modo programmatico e l'integrazione con applicazioni esterne.

Infine, è stato ritenuto necessario il supporto a sorgenti dati eterogenee, includendo sia database di tipo SQL sia NoSQL, al fine di garantire flessibilità e adattabilità a diversi scenari applicativi presenti e futuri.

Ogni requisito analizzato sarà identificato univocamente da una sigla del tipo **R**[**Tipo**][**Priorità**][**Codice**] nella quale:

- **R** sta per Requisito;
- [**Tipo**] può essere:
  - **F** se Funzionale;
  - **NF** se Non Funzionale;
  - **Q** se di Qualità;
  - **V** se di Vincolo.
- [**Priorità**] può essere:
  - **O** per Obbligatorio;
  - **D** per Desiderabile;

  - **P** per Opzionale.
- **[Codice]**: identifica univocamente i requisiti per ogni tipologia. È un numero intero progressivo univoco assegnato in ordine di importanza se il requisito non ha padre, se invece si tratta di un sotto-requisito segue il formato **[Codice padre].[Numero figlio]** e trattandosi di una struttura ricorsiva non c'è limite alla profondità della gerarchia. I codici sono numerati in base alla sezione e alla classificazione (Es: RFO1 = Requisito Funzionale Obbligatorio 1, RNFO1 = Requisito Non Funzionale Obbligatorio 1, RQO1 = Requisito di Qualità Obbligatorio 1, RVO1 = Requisito di Vincolo Obbligatorio 1).

2.1.1. Requisiti funzionali

I requisiti funzionali descrivono cosa deve fare il sistema. Sono le funzionalità concrete che la soluzione deve offrire per raggiungere gli obiettivi del progetto.

Di seguito la tabella con elencati i requisiti funzionali individuati:

Codice	Descrizione	Classificazione
RFO1	Il sistema deve fornire un editor grafico per consentire la creazione e la progettazione visuale dei report	Obbligatorio
RFO2	Il sistema deve permettere l'integrazione con servizi e applicazioni che espongono dati tramite API in formato JSON	Obbligatorio
RFO3	Il sistema deve poter gestire report complessi e strutturati in modo modulare, in particolare elementi di sottoreport	Obbligatorio
RFO4	Il sistema deve consentire l'esportazione dei report in molteplici formati (PDF, DOC, XLSX)	Obbligatorio
RFO5	Il sistema deve consentire l'invocazione programmatica dei report da applicazioni esterne tramite API	Obbligatorio
RFO6	Il sistema deve garantire supporto a database SQL e NoSQL	Obbligatorio
RFO7	Il sistema deve fornire la possibilità di personalizzazione e scripting dei report	Obbligatorio

2.1.2. Requisiti non funzionali

I requisiti non funzionali definiscono come il sistema deve comportarsi, cioè le sue proprietà di qualità interna. Non aggiungono nuove funzioni, ma impongono vincoli di prestazioni, sicurezza, disponibilità, scalabilità, affidabilità e manutenibilità.

Di seguito la tabella con elencati i requisiti non funzionali individuati:

Codice	Descrizione	Classificazione
RNFO1	Il sistema deve supportare la scalabilità, consentendo la gestione di un numero crescente di richieste e utenti	Obbligatorio
RNFO2	Il sistema deve garantire la corretta generazione dei report in modo consistente	Obbligatorio
RNFO3	Il sistema deve consentire una facile integrazione con applicazioni esistenti sviluppate in Java, .NET e Python	Obbligatorio
RNFO4	Il sistema deve garantire tempi di risposta accettabili anche su dataset consistenti	Obbligatorio
RNFD5	Il sistema deve risultare mantenibile, facilitando aggiornamenti, configurazioni e interventi evolutivi	Desiderabile

2.1.3. Requisiti qualitativi

I requisiti qualitativi specificano le proprietà qualitative che influenzano l'esperienza d'uso e la manutenibilità. Si concentrano su aspetti percepibili, come semplicità, chiarezza, flessibilità o estendibilità.

Di seguito la tabella con elencati i requisiti qualitativi individuati:

Codice	Descrizione	Classificazione
RQO1	Il sistema deve fornire un'interfaccia utente intuitiva e di facile utilizzo	Obbligatorio
RQD2	Il sistema deve garantire coerenza nella progettazione e nel layout dei report	Desiderabile
RQD3	Il sistema deve offrire una buona esperienza di sviluppo, supportata da documentazione chiara e API ben definite	Desiderabile

2.1.4. Requisiti di vincolo

I requisiti di vincolo impongono limitazioni o condizioni esterne al progetto: ambienti, tecnologie, compatibilità, strumenti, standard aziendali o legali.

Di seguito la tabella con elencati i requisiti di vincolo individuati:

Codice	Descrizione	Classificazione
RVO1	Il sistema deve poter essere installato e gestito su server locali, senza dipendere esclusivamente da soluzioni cloud	Obbligatorio
RVO2	Il sistema deve prevedere condizioni di licensing sostenibili, evitando vincoli restrittivi legati a licenze proprietarie	Obbligatorio
RVO3	Il sistema deve essere compatibile con l'infrastruttura e lo stack tecnologico aziendale esistente	Obbligatorio
RVD4	Il sistema dovrebbe utilizzare tecnologie consolidate e ampiamente supportate dalla community o dal vendor	Desiderabile

2.1.5. Riepilogo requisiti

Di seguito la tabella riassuntiva dei i requisiti individuati:

Tipologia	Obbligatorio	Desiderabile	Opzionale	Totale
Funzionali	7	0	0	7
Non Funzionali	4	1	0	5
Qualitativi	1	2	0	3
Di Vincolo	3	1	0	4
Totale	15	7	0	22

2.2. Studio della documentazione ufficiale

Per poter individuare soluzioni adeguate è stato effettuato uno studio della documentazione ufficiale delle soluzioni di reportistica trovate, con l'obiettivo di valutarne l'idoneità rispetto ai requisiti funzionali, non funzionali e di vincolo individuati nella fase di analisi.

Per ciascuna piattaforma sono state analizzate le informazioni disponibili nella documentazione ufficiale, includendo guide di installazione, manuali utente, riferimenti API ed esempi di utilizzo. Questa fase ha consentito di acquisire una visione preliminare delle funzionalità offerte, dell'architettura del sistema e delle modalità di integrazione con applicazioni esterne.

Particolare attenzione è stata posta alla disponibilità di un editor grafico per la progettazione dei report, al supporto per sorgenti dati eterogenee (database SQL e NoSQL, formati JSON) e alle modalità di generazione dei report tramite API. Sono stati inoltre valutati gli aspetti relativi al deploy della piattaforma, verificando la possibilità di installazione su server locali e l'eventuale dipendenza da soluzioni cloud.

Lo studio della documentazione ha incluso anche una valutazione preliminare dell'esperienza di sviluppo, analizzando la chiarezza della documentazione, la presenza di esempi e la maturità delle API fornite. Infine, sono stati considerati gli aspetti di licensing, al fine di individuare eventuali limitazioni o vincoli incompatibili con il contesto aziendale.

Sulla base delle informazioni raccolte, è stato possibile effettuare una prima selezione delle soluzioni più promettenti, individuando quelle che soddisfacevano i requisiti obbligatori e che risultavano maggiormente adatte a essere approfondite attraverso la realizzazione di un Proof of Concept.

Lo studio della documentazione ufficiale ha permesso di ridurre il rischio di implementazione, orientando la scelta verso soluzioni coerenti con i requisiti aziendali prima di procedere alla fase di sperimentazione pratica.

2.3. Ricerca di soluzioni alternative

A seguito della definizione dei requisiti e dello studio preliminare della documentazione ufficiale, è stata avviata un'attività di ricerca di soluzioni alternative a JasperServer, con l'obiettivo di individuare piattaforme di reportistica centralizzata potenzialmente idonee al contesto aziendale.

La ricerca è stata condotta consultando la documentazione ufficiale dei principali strumenti di reportistica e Business Intelligence, nonché articoli tecnici, confronti di mercato e risorse provenienti da community open source. Particolare attenzione è stata posta all'individuazione di soluzioni attivamente mantenute e supportate, in grado di garantire continuità nel tempo.

Nel corso di questa fase sono stati adottati criteri di selezione preliminari basati sulla copertura dei requisiti obbligatori individuati, sulla possibilità di deploy su server locali e sull'assenza di vincoli di licensing incompatibili con il contesto aziendale. Le soluzioni che non soddisfacevano tali criteri sono state escluse già in questa fase, riducendo il numero di candidati da analizzare in modo approfondito.

L'esito della ricerca ha portato all'individuazione di un insieme ristretto di soluzioni potenzialmente idonee, successivamente sottoposte a uno studio più dettagliato e a una valutazione comparativa, al fine di selezionare le piattaforme più adatte alla realizzazione del Proof of Concept.

Sulla base di questa analisi preliminare, alcune soluzioni sono state escluse in quanto non soddisfacevano i requisiti obbligatori definiti, mentre altre sono state ritenute idonee per un'analisi più approfondita.

2.4. Setup ambiente di test

Al fine di garantire un confronto coerente e significativo tra le diverse soluzioni di reportistica selezionate, è stato predisposto un ambiente di test controllato e uniforme per l'esecuzione delle soluzioni individuate.

Le soluzioni sono state installate e configurate in ambienti locali, privilegiando modalità di deploy che riflettessero un possibile scenario di utilizzo aziendale. Ove possibile, è stata adottata una configurazione analoga in termini di risorse di sistema, sistema operativo e modalità di esecuzione, al fine di ridurre l'impatto di variabili esterne sui risultati dei test.

Per tutte le piattaforme analizzate è stato utilizzato un insieme comune di dati di test, rappresentativo delle tipologie di input previste nei requisiti, includendo sorgenti dati strutturate e dati in

formato JSON. Questo approccio ha consentito di valutare il comportamento delle diverse soluzioni in condizioni comparabili.

Il setup degli ambienti di test ha avuto come obiettivo principale la validazione delle funzionalità richieste e la verifica della fattibilità tecnica delle soluzioni selezionate, rimandando alle sezioni successive la descrizione dettagliata delle configurazioni adottate e l'analisi delle prestazioni.

Per il setup degli ambienti di test sono state utilizzate tecnologie standard per garantire la riproducibilità e l'isolamento delle piattaforme. In particolare, Docker è stato impiegato per creare ambienti containerizzati uniformi, Postman per testare le API dei server di reportistica e database come MySQL e MongoDB sono stati configurati come sorgenti dati comuni per tutte le piattaforme. I test sono stati eseguiti sul sistema operativo Windows.

2.5. Test funzionali

La presente sezione descrive i test funzionali eseguiti sulle soluzioni individuate, con l'obiettivo di verificare la reale disponibilità e utilizzabilità delle funzionalità richieste, in relazione ai requisiti definiti nella fase di analisi.

2.5.1. Test editor grafico per la creazione dei report

È stata verificata la disponibilità di un editor grafico per la creazione dei report. BIRT mette a disposizione un ambiente di sviluppo simile a JasperStudio, che consente la progettazione visuale dei report. Report Server, invece, non fornisce un editor grafico integrato.

2.5.2. Test supporto input dati JSON

Il supporto ai dati in formato JSON non risulta nativo. In BIRT è possibile utilizzare dati JSON esclusivamente tramite script personalizzati, appositamente realizzato per un uso generale.

2.5.3. Test gestione sottoreport

I test effettuati hanno evidenziato l'assenza di una funzionalità chiara e documentata per la gestione dei sottoreport, rendendo difficile replicare il comportamento offerto da JasperStudio.

2.5.4. Test deploy e gestione su server locale

Sebbene la documentazione ufficiale indichi la possibilità di gestione dei report su server locale, i test pratici hanno evidenziato limitazioni che ne impediscono l'utilizzo effettivo per i casi d'uso richiesti.

2.5.5. Test esportazione report

L'esportazione dei report in diversi formati è risultata generalmente disponibile e non rappresenta un elemento critico nella scelta della soluzione.

2.5.6. Test chiamate API per generazione report

Le chiamate API risultano disponibili, ma la generazione dei report non include correttamente le parti dinamiche basate su script per quanto riguarda alcune soluzioni come Report Server

2.6. Test prestazionali

I test prestazionali sono stati eseguiti con l'obiettivo di valutare i tempi di generazione dei report e il comportamento delle soluzioni analizzate al variare della dimensione dei dati in ingresso. I test non hanno avuto lo scopo di simulare un ambiente di produzione, ma di fornire indicazioni preliminari sulle prestazioni e sulla scalabilità delle soluzioni considerate.

Sono stati definiti diversi scenari di test, variando la dimensione dei dataset in formato JSON e confrontando l'esecuzione dei report in ambiente locale e server. Per ciascuno scenario sono stati misurati i tempi di risposta e osservato l'utilizzo delle risorse di sistema.

2.7. Analisi dei risultati

In questa fase vengono analizzati i risultati ottenuti dai test funzionali e prestazionali condotti sulle soluzioni selezionate. L'analisi ha l'obiettivo di evidenziare la capacità delle piattaforme di soddisfare i requisiti individuati, nonché le eventuali criticità emerse durante la sperimentazione pratica.

2.8. Realizzazione PoC

La fase di realizzazione di un Proof of Concept (PoC) è stata condotta con l'obiettivo di verificare, tramite sperimentazione pratica, la reale capacità delle soluzioni selezionate di soddisfare i requisiti individuati nella fase di analisi. In particolare, il PoC è stato finalizzato a valutare la fattibilità dell'adozione di una soluzione open-source in grado di sostituire JasperServer nel contesto aziendale. E' stato scelto di realizzare un applicativo sviluppato in linguaggio Java che dimostri la possibilità di generare report in locale con librerie open-source individuate in precedenza.

3. Soluzioni analizzate

A seguito dell'attività di ricerca e dello studio preliminare delle soluzioni disponibili, sono state individuate diverse piattaforme di reportistica potenzialmente alternative a JasperServer. La tabella seguente riassume le soluzioni analizzate, evidenziandone la tipologia e i principali motivi di inclusione o esclusione rispetto ai requisiti individuati.

Soluzione	Tipologia	Motivo di inclusione	Limiti principali
BIRT + Knowage	Open-source	Combina editor grafico e server di gestione report; copre gran parte dei requisiti funzionali	Sottoreport non supportati in modo chiaro; Supporto JSON solo tramite script manuali; upload report non automatizzato
DynamicPDF	Commerciale	Buon supporto alla generazione PDF e ai sottoreport	Output limitato al PDF; assenza di API di generazione; supporto JSON limitato
FineReport	Commerciale	Editor grafico avanzato; server locale; diversi formati di output	Mancanza di API; JSON solo locale e tramite scripting; assenza di sottoreport
Helical Insight	Commerciale	Soluzione completa lato report e server; buona facilità d'uso	Versione open-source non più supportata; assenza di API; sottoreport non disponibili
Jsreport	Open-source	Forte supporto API; JSON nativo; server locale; diversi formati di output	Assenza di editor grafico visuale; report da definire in HTML; sottoreport non supportati
Pentaho Report Designer + Server	Commerciale	Ambiente di design maturo; supporto sottoreport; server locale	JSON solo tramite file locale; assenza di API di generazione
Report Server	Open-source e Commerciale	Piattaforma di gestione centralizzata dei report; supporto a diversi formati	Non include editor grafico; copre solo la parte server
Stimulsoft Designer + Server	Commerciale	Piattaforma quasi completa; editor grafico avanzato; supporto JSON e sottoreport	API non native; installazione server limitata (Windows / Docker)
SAP Crystal Report	Commerciale	Editor grafico completo; supporto sottoreport; ampia diffusione	Solo ambiente Windows; JSON non nativo; non include server di gestione

Sulla base di questa analisi preliminare, alcune soluzioni sono state escluse in quanto non soddisfacevano i requisiti obbligatori definiti, mentre altre sono state ritenute idonee per un'analisi più approfondita tramite la realizzazione di un Proof of Concept. In particolare le soluzioni escluse sono state le seguenti: DynamicPDF, FineReport, Helical Insight, Jsreport, Pentaho Report Designer + Server, Stimulsoft Designer + Server, SAP Crystal report. Quelle sottoposte a un'analisi più appro-

fondita sono Birt + Knowage, Report Server e Stimulsoft Designer + server in quanto rispettano un maggior numero di requisiti obbligatori.

4. Confronto tra JasperServer e le alternative

La presente sezione ha l'obiettivo di confrontare JasperServer con le soluzioni alternative analizzate al fine di valutarne la capacità di soddisfare i requisiti funzionali e tecnici individuati. JasperServer è stato utilizzato come riferimento, in quanto adottato in azienda e ritenuto baseline per il confronto.

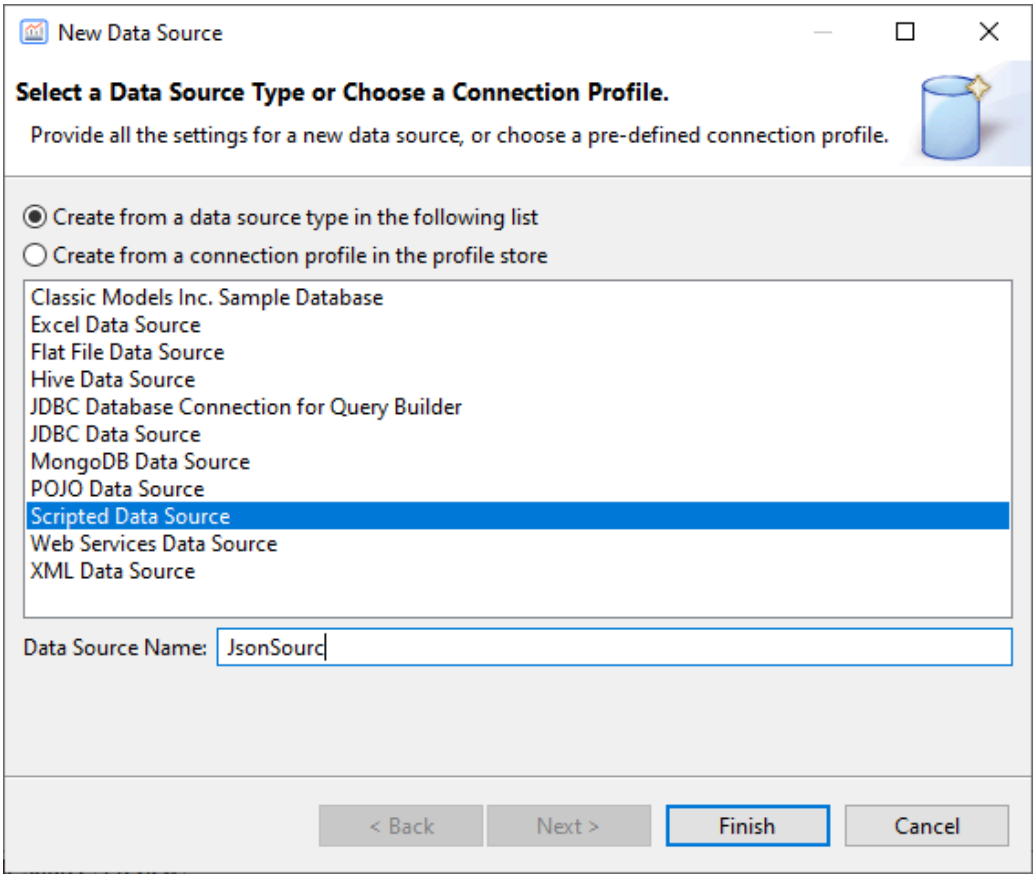
4.1. Birt Design Report

Il prodotto Birt Design Report è un editor per report realizzato sulla piattaforma Eclipse che permette di realizzare report tramite un editor grafico in modo similare a quanto offerto da JasperStudio ed è stato considerato per la parte di realizzazione di report in locale tramite editor grafico.

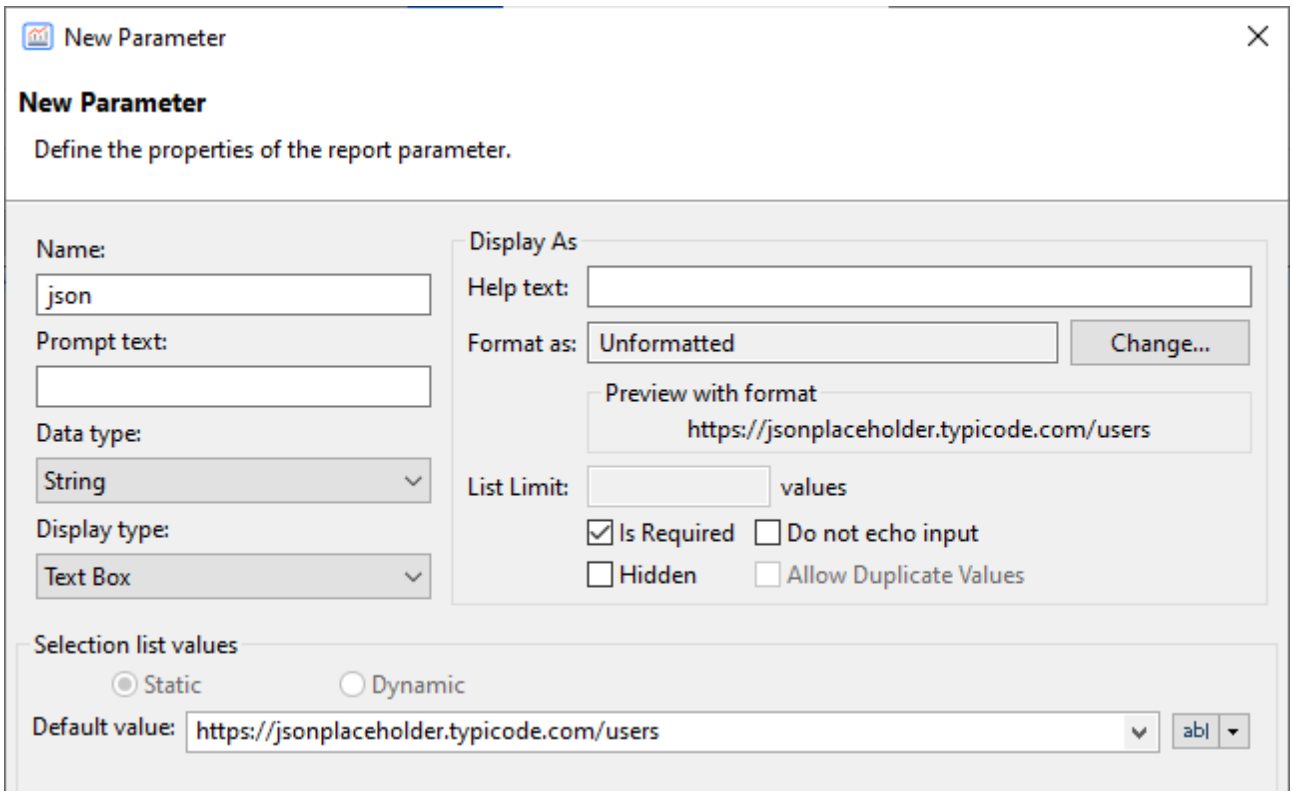
Viene offerto in forma open-source all'indirizzo: https://download.eclipse.org/birt/updates/release/latest attualmente in versione 4.22.0 ma per l'analisi è stata considerata la versione 4.21.0 in quanto era quella disponibile in quel momento.

L'analisi nella sua prima fase si è concentrata nel verificare la presenza e la possibilità di utilizzare dati passati in formati JSON e forniti tramite parametro per popolare il report. Da questa analisi è emerso che per poter replicare questa funzionalità non presente nativamente è necessario definire uno script in Javascript che dato in input un parametro di tipo stringa lo utilizzi per scaricare i dati in formato JSON successivamente utilizzabili per riempire il report. Per realizzare questo si è proceduto in questo modo:

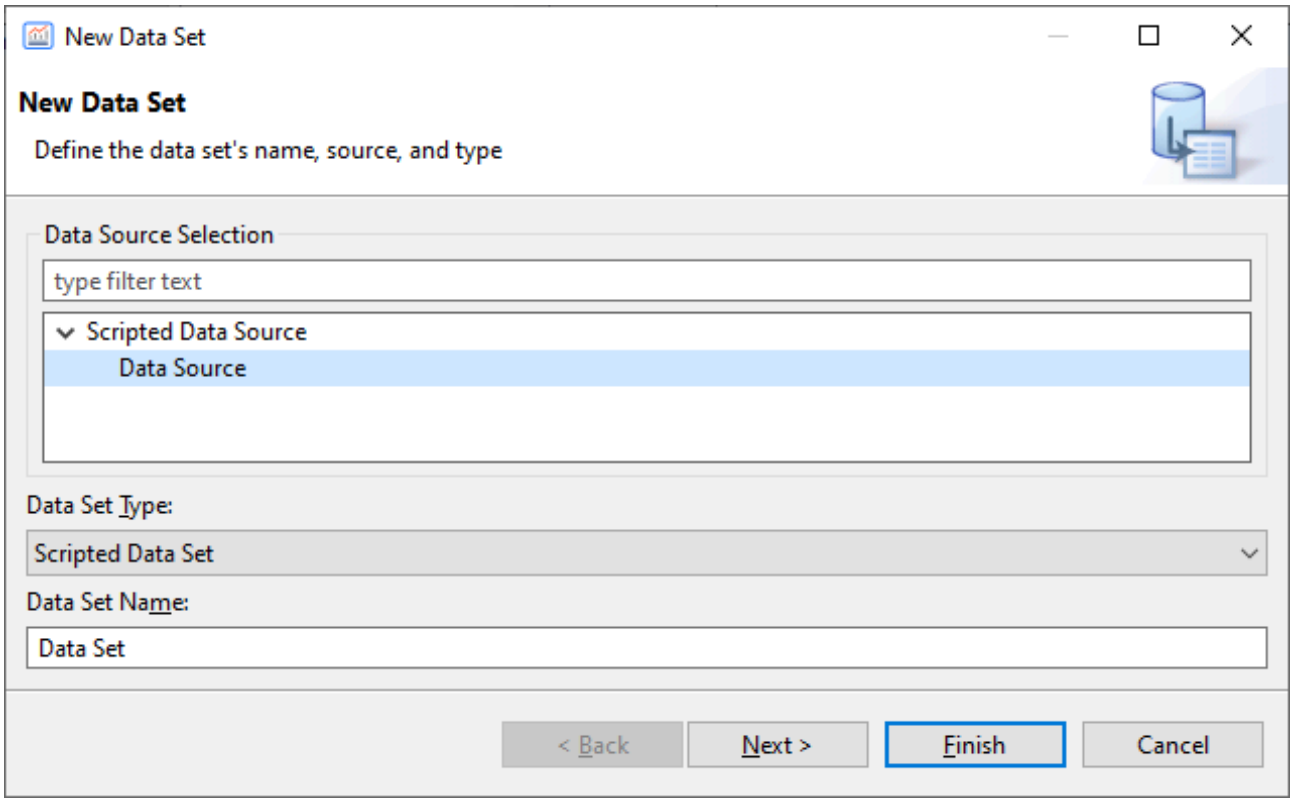
- Definire un data source di tipo Scripted Data Source



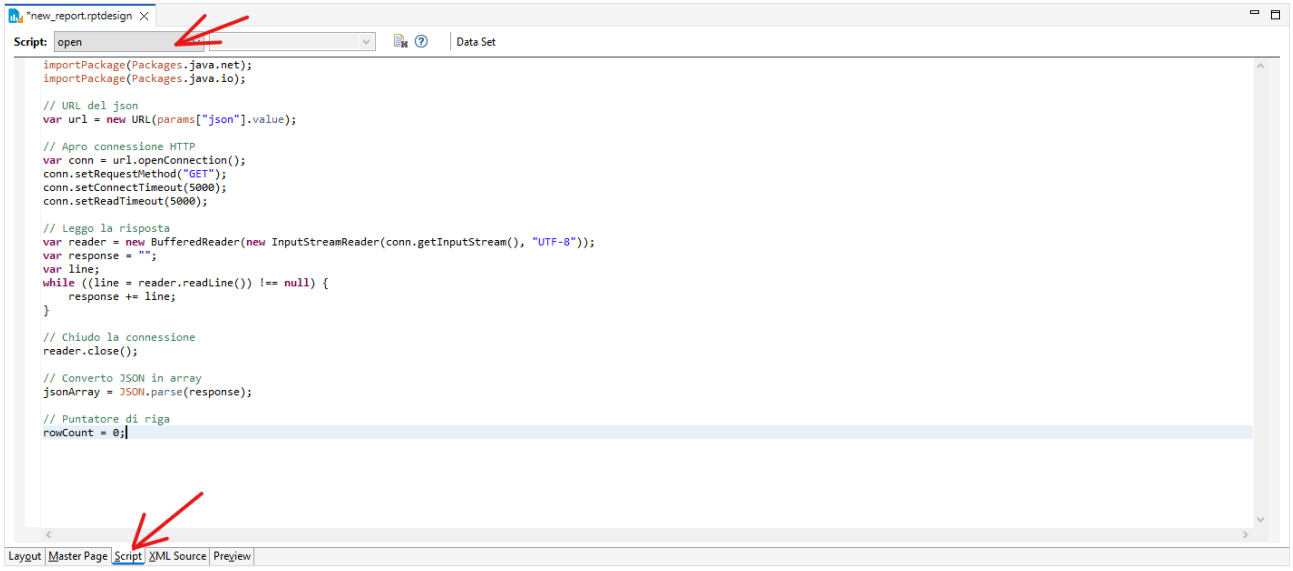
- Aggiungere un parametro, che chiameremo json, in Report Parameters



- Creare un dataset collegandolo al datasource precedentemente



- Clicco sul dataset appena creato e nel file di design seleziono `Script` e successivamente in alto seleziono `open`



A questo punto aggiungo il seguente codice:

Javascript

```
importPackage(Packages.java.net);
importPackage(Packages.java.io);

var urlString = params["json"].value;

if (urlString.startsWith("http") || urlString.startsWith("https")) {
  var url = new URL(urlString);
  var conn = url.openConnection();
  conn.setRequestMethod("GET");
  conn.setConnectTimeout(5000);
  conn.setReadTimeout(5000);

  // Uso StringBuilder per concatenazione efficiente
  var sb = new java.lang.StringBuilder();
  var reader = new BufferedReader(new InputStreamReader(conn.getInputStream(),
"UTF-8"));
  var line;

  while ((line = reader.readLine()) !== null) {
      sb.append(line);
  }

  reader.close();
  conn.disconnect();

  jsonArray = JSON.parse(sb.toString());
}else{
  jsonArray = JSON.parse(urlString);
}

rowIndex = 0;
// Pre-calcolo i campi appiattiti UNA SOLA VOLTA
// Questo evita di ripetere l'operazione per ogni riga
campiAppiattiti = [];

function appiattisciOggetto(obj, prefisso) {
    var risultato = {};
```

```
    prefisso = prefisso || "";

    for (var k in obj) {
        if (!obj.hasOwnProperty(k)) continue;

        var newKey = prefisso ? prefisso + "." + k : k;
        var val = obj[k];

        if (val === null || val === undefined) {
            risultato[newKey] = null;
        } else if (typeof val === 'object') {
            if (Array.isArray(val)) {
                // Converti array in stringa JSON
                risultato[newKey] = JSON.stringify(val);
            } else {
                // Oggetto nested: ricorsione
                var nested = appiattisciOggetto(val, newKey);
                for (var nk in nested) {
                    risultato[nk] = nested[nk];
                }
            }
        } else {
            // Valore primitivo
            risultato[newKey] = val;
        }
    }
    return risultato;
}
// Pre-appiattisco tutti gli oggetti
for (var i = 0; i < jsonArray.length; i++) {
    campiAppiattiti[i] = appiattisciOggetto(jsonArray[i]);
}
// Libero memoria del JSON originale
jsonArray = null;
```

Poi in `fetch` inserisco questo codice:

Javascript

```
if (rowIndex >= campiAppiattiti.length) {
    return false;
}
// Recupero l'oggetto già appiattito
var oggetto = campiAppiattiti[rowIndex];
// Assegno i campi alla riga
for (var campo in oggetto) {
    if (oggetto.hasOwnProperty(campo)) {
        row[campo] = oggetto[campo];
    }
}
rowIndex++;
return true;
```

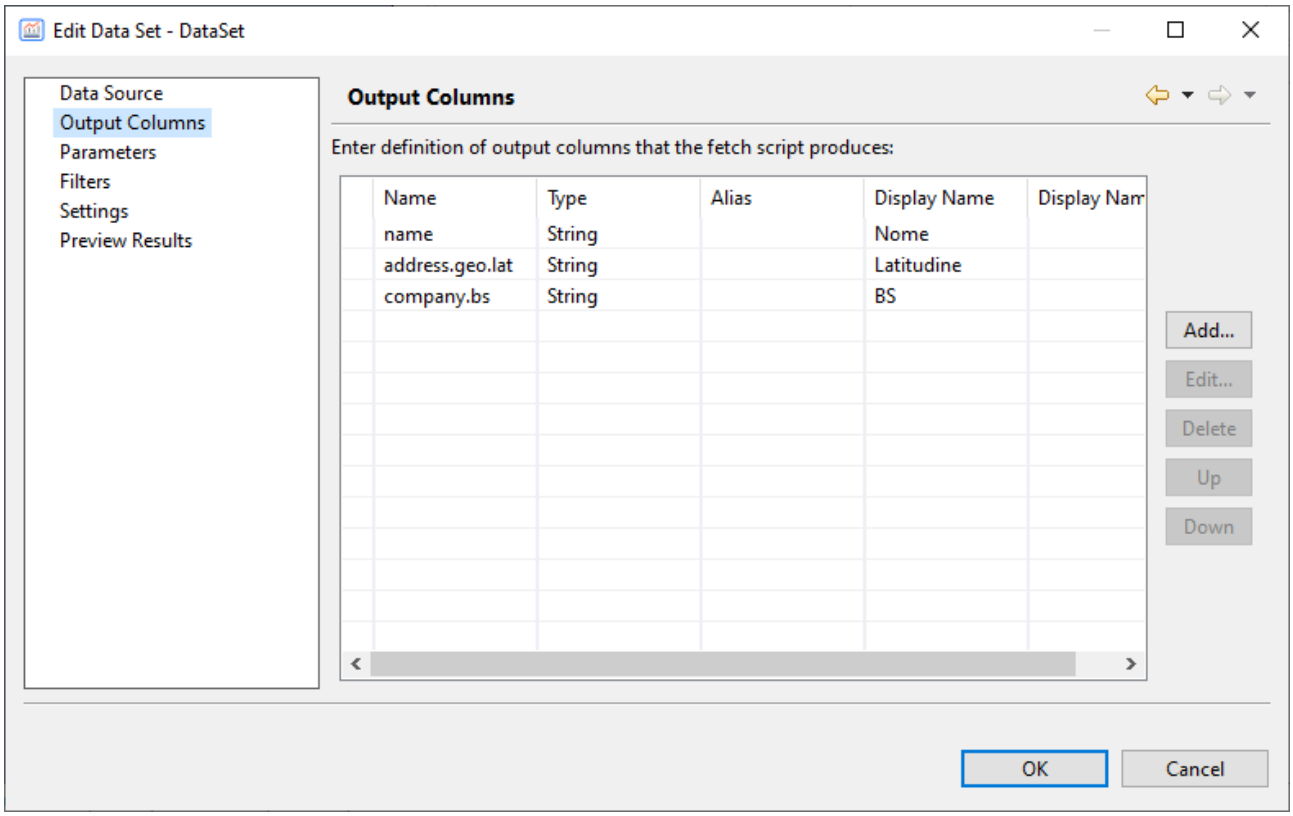
Ed infine su `close` questo codice:

Javascript

```
// Libero la memoria
campiAppiattiti = null;
jsonArray = null;
```

Questo codice, diviso nelle varie fasi di elaborazione del report, permette di recuperare i dati di un file json nel caso in cui al parametro venga passato il percorso del file json mentre scarica i dati in formato json da una sorgente API nel caso gli venga passato un indirizzo. Successivamente chiama una funzione che «appiattisce» l'oggetto recuperato in modo che sia utilizzabile da Birt e che l'utente possa chiamare i campi del json nel formato `padre.figlio` così come veniva fatto in JasperStudio.

Ora è possibile inserire i campi che si vogliono utilizzare nel report cliccando con il tasto destro su `edit` e inserendo i campi nella sezione `Output Columns` nel formato `padre.figlio.nipote.ecc` senza limiti di livello. Per vedere se tutto funziona basta cliccare su `Preview` nella sezione del file di design.



A seguito di questa prima analisi si è passati alla verifica della possibilità di deploy su un server web arbitrario del file di report disegnato così come viene fatto in JasperStudio. Ne è emerso che non è possibile utilizzare questa funzionalità in quanto non è presente o previsto alcun plugin che esegua questo compito. Non esiste infatti alcun riferimento nella documentazione di Birt che faccia riferimento a questa possibilità.

Successivamente si è approfondita l'analisi della possibilità di gestire i sottoreport, ovvero un file di design report contenuto all'interno di un altro file di design report. Ne è emerso che non esiste un elemento grafico fornito da Birt Report Design che permetta l'implemetazione di tale funzionalità.

4.1.1. Conclusioni

A termine di tutta l'analisi eseguita è possibile concludere che il prodotto rispetta tutti i requisiti funzionali richiesti per sostituire il prodotto JasperStudio nella realizzazione di report locali ad eccezione della funzionalità di gestione di sottoreport in quanto è stato verificato che non è presente un elemento ben distinto che gestita tale funzionalità come in JasperStudio.

4.2. Knowage Community Edition

Questo prodotto è stato considerato per la parte server per replicare le funzionalità di JasperServer in quanto mette a disposizione un'implementazione grafica per ospitare report realizzati con Birt. Per poterlo analizzare è stato scelto di utilizzare un'immagine docker disponibile al seguente indirizzo:

https://hub.docker.com/r/knowagelabs/knowage-server-docker che fornisce il prodotto in versione 9.0, ed anche l'installazione locale disponibile all'indirizzo https://www.knowage-suite.com/site/knowage-download/#installer che fornisce il prodotto in versione 8.1.7 . Per poter scaricare l'installer per l'installazione in locale è richiesta la registrazione al sito con un'email e una password e per questa analisi ne è stato inventato uno casuale.

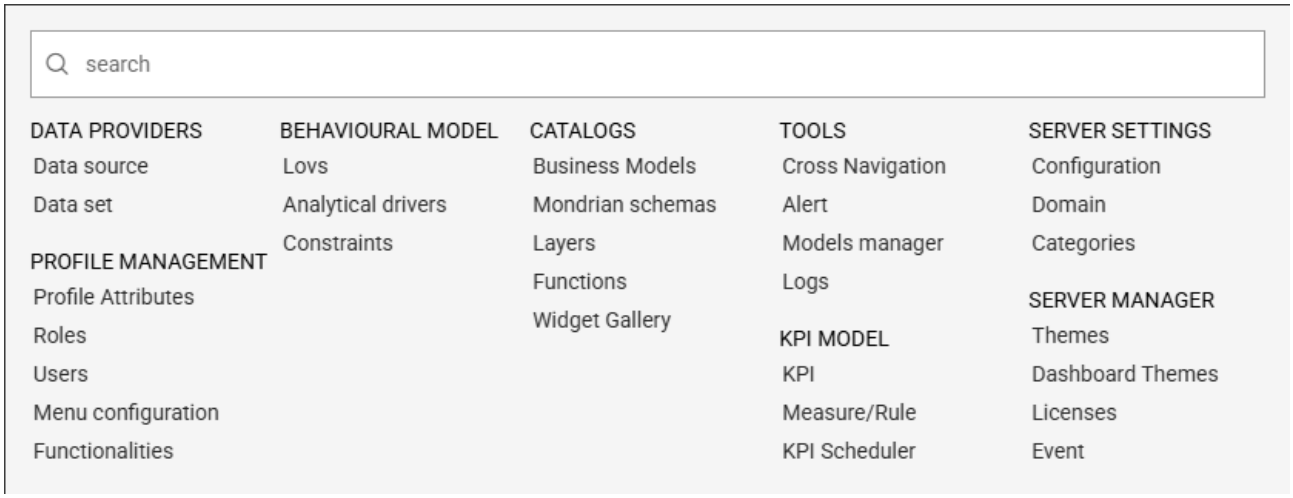
4.2.1. Knowage via Docker

Per poter eseguire Knowage via docker è necessario definire un file `docker-compose.yml` con il seguente codice ed eseguirlo con il comando `docker-compose up` nella stessa cartella del file `.yml`:

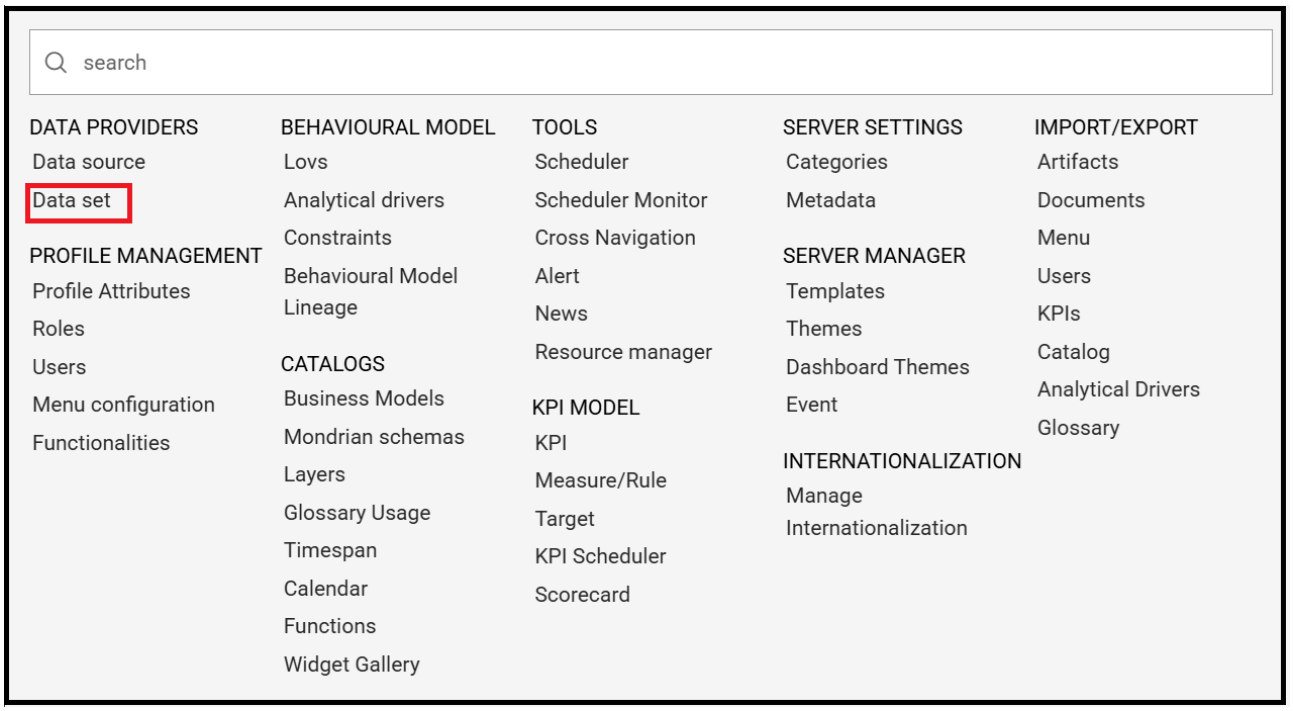
dockerfile

```
networks:
  mynet:
    external: true
services:
  knowage:
    image: knowagelabs/knowage-server-docker:9.0
    container_name: knowage
    depends_on:
      - mysql
    environment:
      DB_HOST: mysql
      DB_PORT: 3307
      DB_DB: knowage
      DB_USER: root
      DB_PASS: root
      HMAC_KEY: root
      CACHE_DB_HOST: mysql
      CACHE_DB_PORT: 3307
      CACHE_DB_DB: knowage_cache
      CACHE_DB_USER: root
      CACHE_DB_PASS: root
      PASSWORD_ENCRYPTION_SECRET: root
      SENSIBLE_DATA_ENCRYPTION_SECRET: root
    networks:
      - mynet
    ports:
      - "8095:8080"
  mysql:
    image: mysql:5.7
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: knowage
    networks:
      - mynet
    ports:
      - "3307:3307"
```

Fatto ciò, basta recarsi all'indirizzo http://localhost:8095/knowage-vue e accedere con le credenziali: user: *biadmin* – password: *biadmin*.
Al suo interno, recandosi nelle impostazioni, si avranno le seguenti opzioni disponibili:



Già a questo punto si possono notare differenze sostaziali tra la documentazione ufficiale e quelle disponibili nella nostra interfaccia:



Non è chiara la motivazione per cui è presente questa differenza in quanto sono stati seguiti tutti gli step della documentazione ufficiale. All'inizio è stata considerata l'ipotesi che ci fossero differenze tra tipi di versioni fornite in immagine docker che potessero in qualche modo limitare le funzionalità di reporting ma nelle informazioni della documentazione come da immagine seguente è chiaramente esplicitata la presenza completa di questa funzionalità.

Main functionalities

Name	Description	EE only
Virtual Assistant	KNOWAGE provides a virtual assistant that, by integrating AI tools (EngGpt), supports users in performing analyses and navigating information	✅
Online Dashboard	The dashboard is an interactive tool designed for visualizing information retrieved from data sets	
Reporting	The advanced reporting capabilities, enabling users to create, customize, and distribute interactive reports based on diverse data sources.	
OLAP	This function allows users to explore multidimensional data interactively, enabling drill-down, slice-and-dice, and pivot operations for in-depth analysis	
KPI	Knowage enables KPI management by defining, monitoring, and visualizing key performance indicators to track business objectives and performance trends	
Data Preparation	Knowage provides data preparation tools to clean, transform, and enrich raw data, ensuring quality and consistency for advanced analytics and reporting.	✅
Python integration	Knowage supports Python integration, allowing users to execute scripts, apply advanced analytics, and embed custom algorithms directly within the BI environment	
Dossier	Knowage allows users to create dossiers by combining multiple reports and documents into a single, organized and interactive view for comprehensive analysis	✅

4.2.2. Knowage via installazione locale

Per verificare la presenza di questi limiti è stata analizzata la versione installabile in locale tramite eseguibile ed accessibile all'indirizzo http://localhost:8080/knowage-vue con le medesime credenziali. In questo caso la versione fornita è la 8.1.7 ma non presenta sostanziali modifiche che riguardano gli obiettivi della nostra analisi. Dopo un'analisi approfondita eseguita in modalità simile all'installazione via Docker è stato riscontrato che sono presenti gli stessi limiti è quindi non è possibile verificare l'effettivo supporto ai report realizzati in Birt

4.2.3. Knowage Studio

Questo è un programma pensato per creare report in modo simile a Birt. È stato utilizzato da Knowage fino alla versione 6.2 mentre, come chiaramente indicato nel sito https://www.knowage-suite.com/site/knowage-download/#1585731491337-57d984ba-4e75, non è più supportato da Knowage a partire dalla versione 8. A fronte di ciò è stato infatti riscontrato che è esattamente un prodotto del tutto simile a Birt Report Design con l'aggiunta di un plugin proprietario che permette il deploy su un server Knowage in modo diretto e di replicare in modo simile il funzionamento di JasperStudio per questa parte. A seguito di numerose prove si è constatato che non si riesce a connettersi alle versioni più recenti di server Knowage. Per questo non si è provveduto ad ulteriori analisi.

4.2.4. Conclusioni

A seguito di tutta questa analisi è possibile concludere che questo prodotto non è adatto alla nostra ricerca in quanto manca la possibilità di verificare la presenza e l'effettiva possibilità di utilizzare le funzionalità essenziali definite nei requisiti funzionali e quindi è stato scartato.

4.3. Report Server

Report Server è stato considerato per la parte server in quanto è un prodotto in grado di eseguire report sia jasper che Birt. Come nei prodotti precedenti è stata eseguita un'analisi delle funzionalità per verificarne il funzionamento e la possibilità di utilizzo.

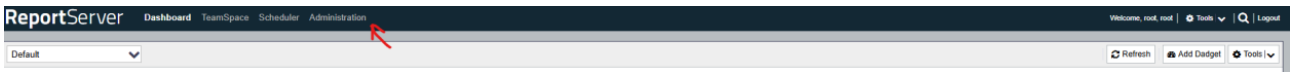
La documentazione ufficiale utilizzata è disponibile al seguente indirizzo: https://reportserver.net/en/index-en mentre il prodotto è possibile scaricarlo alla pagina https://reportserver.net/en/index-en.

Per facilitarne l'analisi è stata utilizzata un'immagine docker reperibile al seguente indirizzo: https://hub.docker.com/r/infofabrik/reportserverenterprise ed implementata con il seguente file `yml`:

dockerfile

```
services:
  mariadb:
    image: mariadb
    container_name: mariadb
    hostname: mariadb
    command: --lower_case_table_names=1
    environment:
      MARIADB_USER: reportserver
      MARIADB_PASSWORD: reportserver
      MARIADB_ROOT_PASSWORD: root
    healthcheck:
      test: "/usr/bin/mysql --user=reportserver --password=reportserver --execute \"SHOW DATABASES;\""
      interval: 10s
      timeout: 30s
      retries: 10
      start_period: 5s
    ports:
      - 3306:3306
  reportserver:
    image: infofabrik/reportserverenterprise:5.0.0
    container_name: reportserver
    hostname: reportserver
    environment:
      REPORTSERVER_DATABASE_HOST: mariadb
      REPORTSERVER_DATABASE_PORT: 3306
      REPORTSERVER_DATABASE_USER: reportserver
      REPORTSERVER_DATABASE_NAME: reportserver
      REPORTSERVER_DATABASE_PASSWORD: reportserver
      REPORTSERVER_DATABASE_ROOTPASSWORD: root
    ports:
      - 8081:8080
    links:
      - "mariadb:mariadb"
    depends_on:
      mariadb:
        condition: service_healthy
networks:
  default:
    name: reportserver-ee-network
```

Ad avvio completato si entra utilizzando le credenziali *(user/password)*: *root/root* . Nella schermata principale bisogna dirigersi sulla scheda `Administrator`, come illustrato nell'immagine:



La quale porta a questa situazione:

Da cui è possibile inserire un report cliccando sull'apposita voce del menù a sinistra, cartella *Report Root->tasto destro->Insert->Birt Report*. A questo punto basta caricare il file `.rptdesign` ed evitare di inserire il datasource in quanto andrebbe in conflitto con quello descritto del file di design appena caricato.

A questo punto, a seguito di numerose prove nel server e di modifiche al file `.rptdesign` l'unico risultato ottenuto è l'esecuzione di un report solo nelle sue parti statiche, ovvero tutte le parti presenti come elementi ben definiti, mentre quelle definite nello script interno del report non vengono eseguite. Questo perché i file di script supportati sono di tipo `groovy`.

Sono state provate anche chiamate API con passaggio di dati tramite parametro e funzionano ma generano un report come nella prova precedente.

Si è quindi scelto di non procedere ulteriormente all'approfondimento di questo prodotto in quanto introduce un ulteriore livello di complessità non desiderato nel convertire lo script da Javascript a Groovy che sarebbe difficilmente implementabile da un'utenza comune quale quella prevista come destinataria di questi prodotti.

4.4. Stimulsoft Designer e Stimulsoft Server

Questa combinazione di prodotti di tipo commerciale è stata analizzata in quanto se utilizzati in combinazione fornisce un'alternativa a JasperStudio e JasperServer. In particolare il prodotto Stimulsoft Design permette di disegnare report mentre il prodotto Stimulsoft Server permette di implementare un server in locale collegabile a Stimulsoft Design per effettuare caricamento di report dal design al server.

Il limite maggiore di questa soluzione è il suo ambiente di utilizzo, rigorosamente Windows, che lo rende da subito non utilizzabile come possibile alternativa a Jasper.

5. Realizzazione e documentazione tecnica configurazione del PoC

A seguito delle analisi e dei confronti effettuati si è scelto di procedere alla realizzazione di un PoC in linguaggio `Java` per un'applicativo locale e in linguaggio `Python` per una versione server utilizzando la soluzione open-source Birt Design Report e le sue librerie Birtruntime che permettono di sviluppare un applicativo in grado di compilare report Birt anche in assenza del programma di design.
La versione di BIRT utilizzata è 4.21 le cui librerie sono disponibili all'indirizzo:
https://download.eclipse.org/birt/updates/release/latest/
E' stato inoltre utilizzato il programma Postman per la creazione di un server mock e personalizzare la risposta a una chimata API.

5.1. Realizzazione di un programma Java per generare report BIRT

Il Proof of Concept è stato realizzato in ambiente locale, utilizzando Eclipse per la fase di sviluppo del codice Java, al fine di garantire facilità di sviluppo e ripetibilità dei test.

Il prodotto è stato sviluppato in queste condizioni:
- Sistema operativo: Windows
- Strumenti: Eclipse, Birt Report Design, librerie BIRT v4.21, Java v.21

Architettura software: Model-View-Controller

Nel model sono state sviluppate le seguenti classi:

- **BirtDesignToDocument**: è una delle classi più importanti e si occupa di gestire tutta la parte di generare documenti a partire da un report Birt in base al formato scelto (PDF, DOC, XLSX, HTML) richiamando le varie funzioni dedicate. Utilizza l'API di Birt `(org.eclipse.birt.report.engine.api)` per aprire, eseguire e renderizzare i report.
- **ReportConfig**: questa classe si occupa di rappresentare la configurazione per la generazione del report, quindi la sorgente del file di design.

Nella view è stata sviluppata tutta la parte grafica relativa a come viene visualizzata l'applicazione alla sua esecuzione nella classe **Window**.

Nel controller è stata implementata la classe **Controller** che gestisce tutti gli input effettuati dall'utente tramite grafica e gestisce la comunicazione tra i model e la view.

5.1.1. BirtDesignToDocument

- Attributi:
  - `sourceJson`: percorso del file JSON da usare come input per i parametri del report (può essere anche un URL).
  - `sourceBirt`: percorso del file .rptdesign contenente il design del report.
  - `json` stringa contenente i dati JSON letti dal file.
- Metodo generateDocument(String format): Accetta un formato di output ed in base a questo chiama il metodo specifico di generazione (generatePDF, …). Ritorna `true` se la generazione ha successo, `false` altrimenti.
- Metodi di generazione per formato : Ogni metodo segue lo stesso schema:
  - Configura BIRT Engine

```Java
EngineConfig config = new EngineConfig();
config.setEngineHome(System.getProperty("user.home")+"/Desktop/BirtPDF");
config.setLogConfig(System.getProperty("user.home")+"/Desktop/BirtPDF",
Level.FINE);
Platform.startup(config);
```

Imposta la cartella di lavoro e il logging e avvia il framework Platform di BIRT.

- Crea il report engine

```Java
IReportEngineFactory factory = (IReportEngineFactory) Platform.createFactoryObject(
    IReportEngineFactory.EXTENSION_REPORT_ENGINE_FACTORY
);
IReportEngine engine = factory.createReportEngine(config);
```

`IReportEngine` permette di aprire il report e di creare task di esecuzione.

- Apre il design del report

```Java
IReportRunnable design = engine.openReportDesign(sourceBirt);
IRunAndRenderTask task = engine.createRunAndRenderTask(design);
```

`IReportRunnable` rappresenta il report design mentre `IRunAndRenderTask` si occupa del task che esegue il report e lo renderizza in un formato finale.

- Gestisce i parametri JSON

```Java
setJsonParameters(engine, design, task);
```

In particolare se `sourceJson` è un URL, imposta il parametro json direttamente altrimenti se è un file locale, legge il contenuto e imposta tutti i parametri String presenti nel design.

- Imposta opzioni di rendering in base alla funzione chiamata, quindi per un PDF: `PDFRenderOption`, per un DOC: `RenderOption`, per un XLSX: `EXCELRenderOption`, per un HTML: `HTMLRenderOption`.
  Infine specifica il nome del file di output, che include un timestamp per evitare conflitti.

- Esegue il task e chiude il motore

```Java
task.run();
task.close();
engine.destroy();
Platform.shutdown();
```

Dopo l'esecuzione, il file viene generato nella cartella `Desktop` dell'utente per comodità.

5.1.2. ReportConfig

La classe ReportConfig è un model di configurazione che rappresenta i parametri necessari alla generazione di un report BIRT. Il suo scopo è incapsulare in un unico oggetto tutte le informazioni richieste per pilotare il processo di generazione del documento (JSON di input, report design, formato di output).

- Attributi della classe

```Java
private String jsonSource;
private String birtFilePath;
private String outputFormat;
private String jsonSourceType;
```

- `jsonSource`: Rappresenta la sorgente dei dati JSON utilizzati dal report. Può essere un URL (API REST) o un percorso a file locale
- `birtFilePath`: Contiene il percorso del file di design `.rptdesign`
- `outputFormat`: Contiene il formato di output scelto (PDF, DOC, XLSX, HTML)

  - `jsonSourceType`: Specifica la tipologia della sorgente `jsonSource`, se come fonte API oppure File
- Costruttore

```Java
public ReportConfig() {
    this.outputFormat = "PDF";
    this.jsonSourceType = "API";
}
```

  Di default il costruttore definisce come formato di output PDF e come sorgente dati API.
- Metodi di accesso
  La classe fornisce i classici metodi di accesso per ogni attributo.
- Metodo di validazione

```Java
public boolean isValidConfig() {
    return jsonSource != null && !jsonSource.isBlank()
        && birtFilePath != null && !birtFilePath.isBlank();
}
```

  Questo metodo verifica che la configurazione sia minimamente valida prima di avviare la generazione del report. In particolare che `jsonSource` e `birtFilePath` non siano nulli o vuoti e ritorna `true` solo nel caso in cui queste due condizioni non si verificano.

5.1.3. Window

La classe `Window` rappresenta la View dell'applicazione desktop per la generazione di report BIRT. Il suo compito è costruire e visualizzare l'interfaccia grafica, fornendo al controller componenti UI (campi di testo, pulsanti, combo box) e metodi di supporto per la visualizzazione di finestre di dialog.

Con il metodo `initialize()` istanzia e configura tutti i componenti e definisce dimensioni, font, tooltip e stato iniziale.

5.1.4. Controller

La classe Controller rappresenta il controller dell'applicazione desktop Swing per la generazione di report BIRT. Il suo compito è coordinare l'interazione tra la View (Window) e il Model (ReportConfig, BirtDesignToDocument), gestendo: eventi dell'interfaccia utente, validazione degli input, aggiornamento della configurazione, avvio asincrono della generazione del report.

- Inizializzazione della classe
  Viene effettuata dal metodo `initController()` che viene chiamato dal costruttore al momento della generazione della classe. Si occupa di collegare le azioni dell'utente (click, selezioni, input testuali) e di attivare la validazione dinamica dei campi.
- Generazione file
  Attraverso i metodi `handleLoadBirt()` e `handleLoadJson()` si occupa gestire la gestione del caricamento di un file di design e di un file json aprendo un `FileDialog` e impostando il `filePath` nel ReportConfig e in Window.
- Gestione tipo sorgente JSON
  Con il metodo `handleJsonTypeChange()` viene gestito il cambiamento di selezione di sorgente da API a FILE e viceversa. In particolare:
  - Abilita/disabilita il pulsante di caricamento file
  - Rende editabile o meno il campo di testo
  - Imposta tooltip contestuali
  - Aggiorna `jsonSourceType` nel model
- Validazione input
  Con il metodo `validateInputs()` viene controllato in tempo reale che la sorgente Json inizi con http nel casi sia stato selezionato API oppure che non sia vuota nel caso in cui sia stato selezionato

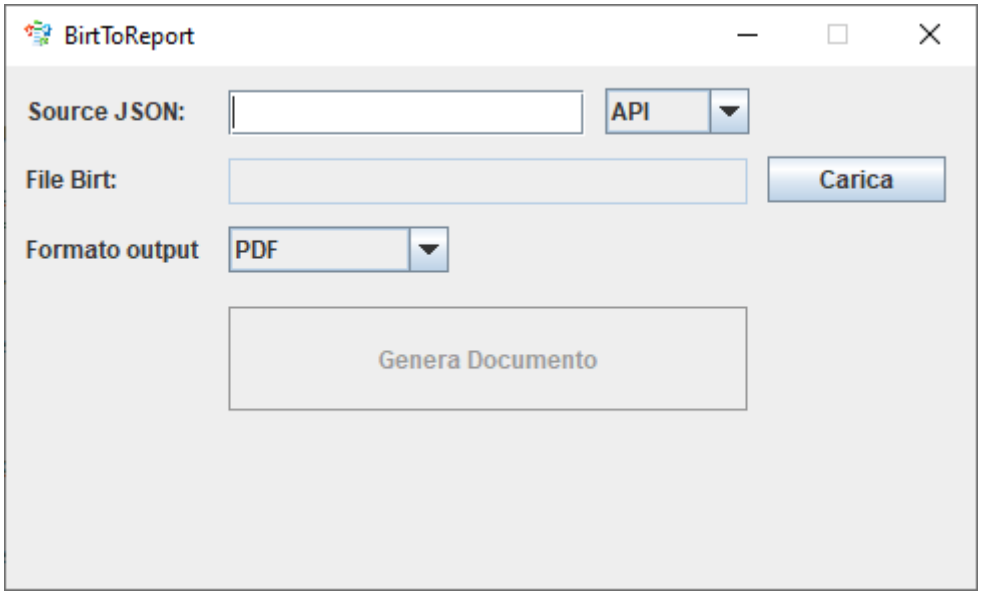
File e che la selezione del file Birt non sia vuota. Se queste 2 condizioni sono valide viene chiamato il metodo

```Java
1 view.getBtnGenerateDocument().setEnabled(true);
```

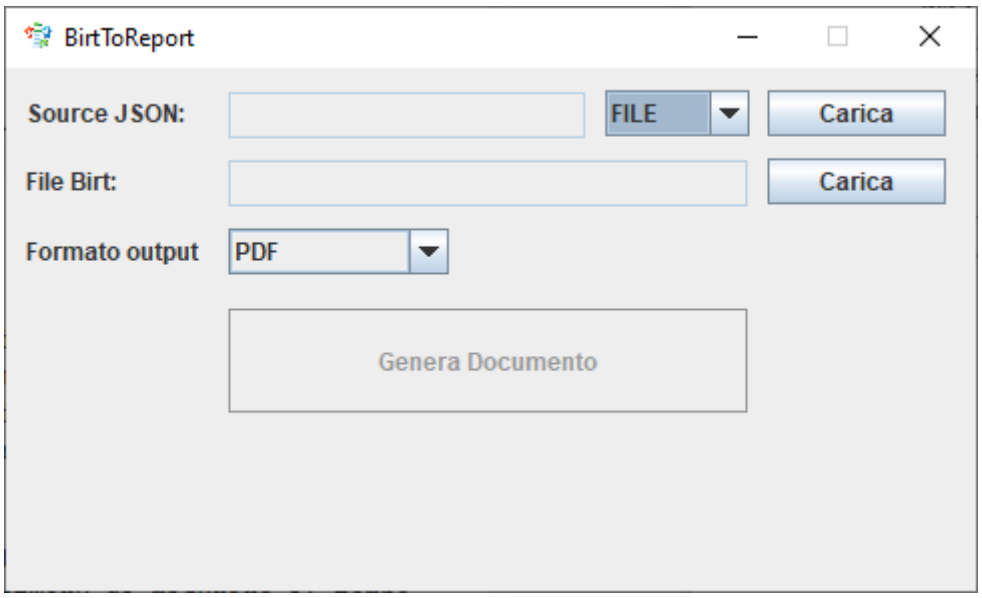
- Generazione del documento
  Viene effettuata dal metodo `handleGenerateDocument()` e svolge diverse funzioni:
  - Aggiorna il model ReportConfig con i valori corretti della sorgente Json e del formato di output;
  - Controlla attraverso la funzione `isValidConfig` che siano presenti e validi i valori `jsonSource` e `birtFilePath`
  - Avvia la generazione del report in modo asincrono attraverso la classe `SwingWorker`, che mi permette di evitare il blocco del thread UI e di migliorare la responsività dell'interfaccia. Al suo interno presenta due metodi:
    - `protected Boolean doInBackground()` che si occupa di richiamare la generazione del report e restituisce `true` solo se è andata a buon fine
    - `protected void done()` che definisce le azioni che vengono eseguite quando il la generazione del report si è conclusa positivamente oppure mostra una finestra di errore se la generazione del report si è conclusa negativamente.
- Selezione file
  Attraverso i metodi `chooseBirtFile()` e `chooseJsonFile()` vengono aperte delle finestre di dialogo per permettere la selezione del file di design `.rptdesign` e del file `.json`. Ritornano il path dei rispettivi file se selezionati oppure null se non sono stati selezionati.

5.1.5. Visualizzazione grafica

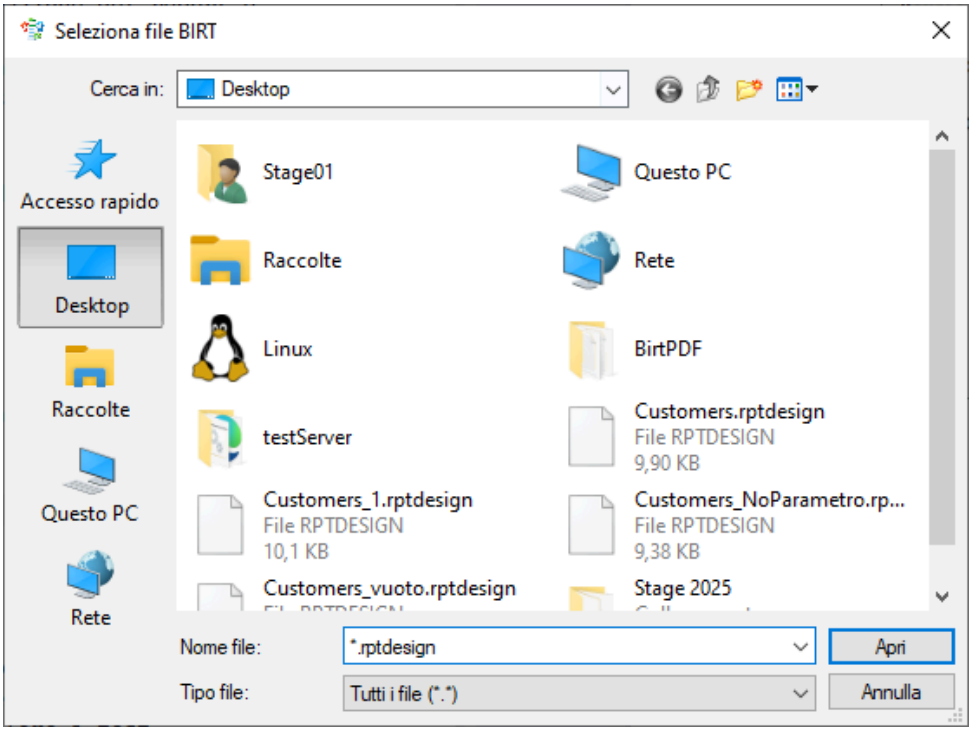
All'avvio l'applicazione si presenta in questo modo:



Nel caso in cui si selezioni come Source Json File la finestra si modifica in questo modo:



Fino a questo punto il pulsante Genera documento resta disabilitato. Quando verranno selezionait i file Birt e il file Json o messa una sorgente API valida il pulsante si abiliterà.
Cliccando su Carica si aprirà una finestra di dialogo per selezionare i file corrispondenti:



5.2. Realizzazione di un report server BIRT

Per realizzare un server in grado di restituire un report compilato dato un un file di design `.rptdesign` e un API json è stata utilizzata come base di partenza il codice java del PoC locale e successivamente è stato sviluppato uno script in python per eseguire un server in locale. Di fatto è un servizio REST sviluppato con Flask che consente la generazione centralizzata di report BIRT a partire da file .rptdesign e da una sorgente dati esposta tramite API JSON. L'obiettivo principale di questa attività è stato quello di:

- simulare il comportamento di un report server centralizzato;
- consentire la generazione di report compilati on-demand;
- riutilizzare il PoC Java precedentemente sviluppato;
- esporre il servizio tramite API REST, rendendolo indipendente dal client.

Il tutto è stato sviluppato in queste condizioni:
- Sistema operativo: Windows
- Strumenti utilizzati: Visual Studio Code, Claude AI, librerie BIRT v.4.21, Java v.21, Flask per python v.3.0.3, Python v.3.14.

Tutte le indicazioni di funzionamento sono state scritte nel file `readme.md` presente nella directory principale del progetto.

5.2.1. Architettura della soluzione

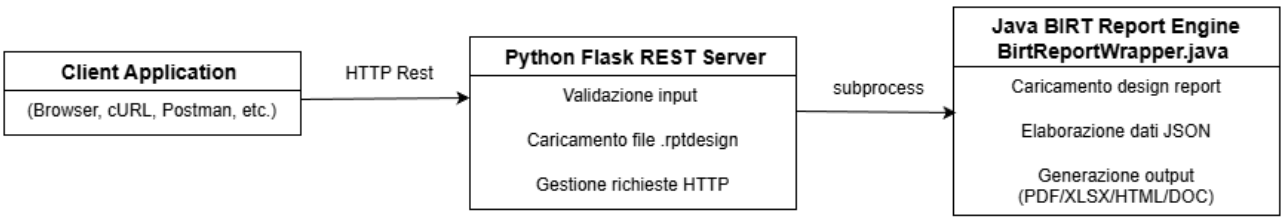
La soluzione sviluppata adotta un'architettura ibrida, composta da due componenti principali:
- Motore di generazione report basato su BIRT Runtime, sviluppato in Java;
- Server REST sviluppato in Python (Flask), responsabile dell'orchestrazione delle richieste.

Componenti principali

- *Java (BIRT Runtime)*
  Gestisce:
  - caricamento del file .rptdesign;
  - recupero dei dati da API JSON;
  - compilazione e generazione del report nel formato richiesto.
- *Python (Flask)*
  Espone un'interfaccia REST che:
  - riceve le richieste HTTP;
  - valida i parametri;
  - invoca il processo Java tramite subprocess;
  - restituisce il file generato al client.
- *Flusso di esecuzione*
  Il client invia una richiesta HTTP POST al server REST.
  Il server riceve:
  - il file .rptdesign;
  - l'URL dell'API JSON;
  - il formato di output richiesto.

  Il server avvia il processo Java di compilazione del report.
  Il report viene generato e salvato su filesystem.
  Il file finale viene restituito come risposta HTTP.

Graficamente può essere rappresentata in questo in questo modo:



Questa architettura consente di separare la logica di business (Java/BIRT) dalla gestione delle richieste (Python).

5.2.2. Implementazione

5.2.2.1. BirtReportEngine

Le classi java sviluppate nel PoC precedentemente sono state riadattate e incluse nel file `BirtReportEngine.java` così definita:

Java

```java
package com.report.model;

public class BirtReportWrapper {
    public static void main(String[] args) {
        // Entry point per esecuzione da linea di comando
    }
}

class BirtDesignToDocument {
    private final String sourceJson;
    private final String sourceBirt;
    private final String outputDir;
    private final String birtHome;

    public String generateDocument(String format) throws Exception {
        // Routing verso metodi specifici per formato
    }
}
```

In particolare nel main vengono controllati che i parametri passati siano corretti e nel formato giusto, vengono definiti i loge viene fatta partire la generazione del report tramite la classe BirtDesignToDocument, già utilizzata nel PoC locale per gestire tutta la parte di generazione del report. Si rimanda a quella sezione per i dettagli di funzionamento.

5.2.2.2. Server.py

Questo componente svolge il ruolo di:

- API Gateway di reporting;
- Orchestratore di processi Java esterni;
- Gestore di file temporanei e output;
- Interfaccia REST standardizzata per la generazione di report.

All'interno dell'architettura complessiva, il server Python si colloca tra:

- Client applicativi (web, backend, test client);
- Motore BIRT incapsulato in un'applicazione Java standalone.

Per impostazione predefinita utilizza questa configurazione

Parametro	Valore di default	Descrizione
BASE_DIR	~/reports	Directory base per runtime
UPLOAD_DIR	~/reports/upload	Directory per file .rptdesign temporanei
OUTPUT_DIR	~/reports/output	Directory per report generati
BIRT_HOME	~/reports/birt	Directory utilizzata da BIRT engine
LOG_DIR	~/reports/logs	Directory utilizzata per i file di log
PORT	5000	Porta del server
MAX_FILE_SIZE	50 MB	Dimensione massima upload
TIMEOUT	300 secondi	Timeout per la generazione di report

E' composto da queste parti principali:

- Qui viene definito dove vengono salvati i log, i report generati e i file utilizzati per far funzionare il server.

Python

```python
# Configurazione
BASE_DIR = Path.home() / "reports"
UPLOAD_DIR = BASE_DIR / "uploads"
OUTPUT_DIR = BASE_DIR / "output"
BIRT_HOME = BASE_DIR / "birt"
LOG_DIR = BASE_DIR / "logs"

# Inizializzazione Flask
app = Flask(__name__)
CORS(app)
```

- Questa funzionone si occupa di far partire la generazione di un report

Python

```python
def generate_birt_report(birt_file_path, json_api_url, output_format):
    # 1. Costruzione classpath
    classpath = f"bin{classpath_sep}{lib_pattern}"

    # 2. Comando Java
    java_cmd = [
        "java", "-cp", classpath,
        "com.report.model.BirtReportWrapper",
        str(birt_file_path), json_api_url,
        str(OUTPUT_DIR), str(BIRT_HOME), output_format
    ]

    # 3. Esecuzione subprocess
    result = subprocess.run(java_cmd, capture_output=True,
                            timeout=300, encoding='utf-8')

    # 4. Parsing output e restituzione path file
    return output_file
```

- In queste parti definisce le API disponibili:
  - Health check: per controllare se il server è in funzione

Python

```python
@app.route('/api/reports/health', methods=['GET'])
def health_check():
```

  - Formats: dice che formati di output supporta il server

Python

```python
@app.route('/api/reports/formats', methods=['GET'])
def get_formats():
```

  - Generazione report via riga di comando

Python

```python
@app.route('/api/reports/generate', methods=['POST'])
def generate_report():
```

  - Pulizia della repository: per pulire le cartelle utilizzate dal server e liberare spazio

Python

```python
@app.route('/api/reports/cleanup', methods=['POST'])
def cleanup_old_files():
```

Esempio di utilizzo in Linux:

Script

```
# Generazione PDF
curl -X POST http://localhost:5000/api/reports/generate \
```

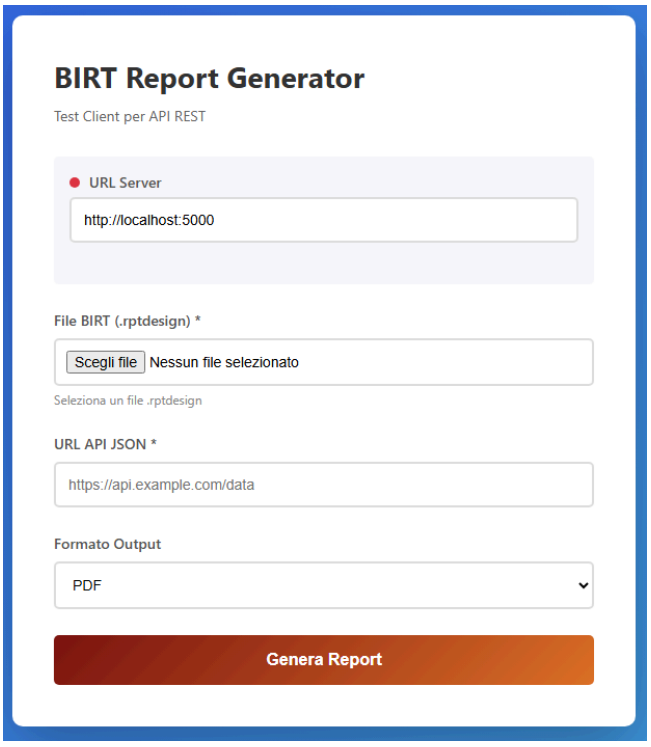
```
  -F "birtFile=@report.rptdesign" \
  -F "jsonApiUrl=https://api.example.com/data" \
  -F "format=PDF" \
  --output report.pdf

# Health check
curl http://localhost:5000/api/reports/health

# Pulizia file vecchi
curl -X POST http://localhost:5000/api/reports/cleanup \
  -H "Content-Type: application/json" \
  -d '{"days": 7}'
```

5.2.2.3. Pagina di test in HTML

E' stata sviluppata una pagina web nel file `test_client.html` per facilitare il test del server nella generazione del report. Presenta questa pagina:



5.2.3. Prestazioni

Durante i test effettuati in ambiente di sviluppo, sono emerse le seguenti considerazioni:

- Tempo medio di generazione:
  5–30 secondi per report di dimensioni medio-piccole;
- Numero di richieste simultanee gestibili:
  circa 5–10, in funzione delle risorse disponibili;
- Consumo di memoria:
  almeno 1–2 GB di RAM liberi per un utilizzo stabile.

Questi valori indicano che la soluzione è adatta a carichi limitati, tipici di ambienti interni o PoC, ma non a scenari enterprise ad alta concorrenza senza ulteriori ottimizzazioni.

5.2.4. Valutazione complessiva

Lo sviluppo di questo Report Server custom dimostra che:

- è tecnicamente possibile utilizzare BIRT come motore server-side;

- l'assenza di un server ufficiale richiede sviluppo custom;
- i costi di manutenzione e complessità crescono rapidamente.

Questa soluzione può essere considerata:

- valida come PoC tecnico;
- utile per ambienti di test o utilizzi limitati;
- non consigliata come sostituto diretto di JasperServer in contesti enterprise senza un investimento significativo.

6. Confronti prestazionali

Per eseguire dei test prestazionali che fossero comparabili tra JasperServer e il PoC si è cercato di mantenere le condizioni iniziali più simili possibili, tenendo conto che il primo è eseguito su un container docker e il secondo con un programma locale, condizione che può aver influito leggermente sui risultati. Non avendo bisogno di dati di precisione elevata si è proceduto a fare i test.

Pe questo primo test di velocità è stato utilizzato un report semplice di poche righe e una fonte Json con dati annidati di 10 elementi. Per quanto riguarda JasperServer è stato utilizzato lo strumento per gli sviluppatori disponibile in `Google Chrome` che permette attraverso la scheda network di misurare il tempo che occorre al server per eseguire la richiesta senza dover obbligatoriamente scaricare il file generato mentre per il PoC è stato utilizzato `VisualVM`, un programma open-source per analizzare processi in esecuzione.
Per JasperServer sulla barra degli indirizzi è stata inserito:
http://localhost:8088/jasperserver/rest_v2/reports/Report/test.pdf?json=https://2cc1de8b-6155-4fad-bf6a-aed4221a4810.mock.pstmn.io/user
Questa richiesta ha portato a questi risultati:

test.pdf?json=https://jsonplace...	200	docume...	Other	0.5 kB	17.98 s
test.pdf?json=https://jsonplace...	200	docume...	Other	0.4 kB	1.19 s
test.pdf?json=https://jsonplace...	200	docume...	Other	0.4 kB	788 ms

Come è possibile notare la prima richiesta ha richiesto circa 18 secondi per essere eseguita mentre le richieste successive sono risultate nettamente più veloci nell'ordine del secondo.

Per il PoC è stato eseguito il test inserendo l'indirizzo del json sulla barra corrispondente e ha portato questi tempi:

SwingWorker-pool-1-thread-1	13.191 ms	(100%)	12.886 ms	(100%)
SwingWorker-pool-1-thread-4	705 ms	(100%)	593 ms	(100%)
SwingWorker-pool-1-thread-2	693 ms	(100%)	693 ms	(100%)
SwingWorker-pool-1-thread-3	603 ms	(100%)	603 ms	(100%)

Come è possibile notare il tempo di esecuzione delle richieste successive alla prima è simile a JasperServer mentre la prima richiesta è stata di qualche secondo più veloce ma questo dato può essere ricondotto al fatto che Docker sul quale gira JasperServer richieda più risorse e quindi sensibile alla macchina sul quale è in funzionamento.

Un secondo test è stato eseguito con un file report più complesso e un json di 100 elementi con i seguenti risultati:

Name	Type	Size	Time
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	16.55 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	1.53 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	994 ms

Name	Total Time	
SwingWorker-pool-1-thread-1	9.497 ms	(100%)
SwingWorker-pool-1-thread-2	1.587 ms	(100%)
SwingWorker-pool-1-thread-3	1.194 ms	(100%)

Un terzo test è stato eseguito con un file report più complesso e un json di 1000 elementi con i seguenti risultati:

Name	Type	Size	Time
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	7.98 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	3.67 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	2.19 s

Name	Total Time	Total Time (CPU)
SwingWorker-pool-1-thread-1	26.903 ms (100%)	26.104 ms (100%)
SwingWorker-pool-1-thread-2	3.195 ms (100%)	2.697 ms (100%)
SwingWorker-pool-1-thread-3	2.792 ms (100%)	2.273 ms (100%)

Ed infine è stato effettuato un test partendo da un report senza passaggio API, che quindi genera un report vuoto, e successivamente con un json di 1000 elementi e questi sono i risultati:

Name	Type	Size	Time
test.pdf	document	0.4 kB	9.65 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	2.46 s
test.pdf?json=https://2cc1de8b-6155-4fad-bf6a...	document	0.4 kB	2.17 s

Name	Total Time	Total Time (CPU)
SwingWorker-pool-1-thread-1	7.600 ms (100%)	7.600 ms (100%)
SwingWorker-pool-1-thread-2	5.674 ms (100%)	4.998 ms (100%)
SwingWorker-pool-1-thread-3	3.504 ms (100%)	2.798 ms (100%)
SwingWorker-pool-1-thread-4	3.191 ms (100%)	1.984 ms (100%)

Con questo test è possibile concludere che i tempi di esecuzione di un report sui due prodotti è comparabile e non comporta differenze di tempo sostanziali.

I risultati hanno evidenziato un degrado delle prestazioni all'aumentare della complessità dei report e del volume dei dati, in particolare per le soluzioni che richiedono l'esecuzione di script personalizzati per la lettura dei dati anche se non in modo sostanzialmente differente.

6.1. Tabella riassuntiva

Di seguito una tabella riassuttiva dei test prestazionali eseguiti:

Soluzione	Dataset	Tempo di esecuzione	Note
JasperServer	10	18 secondi	Prima esecuzione
JasperServer	10	1 secondo	Seconda esecuzione
PoC	10	13 secondi	Prima esecuzione
PoC	10	700 millisecondi	Seconda esecuzione
JasperServer	100	17 secondi	Prima esecuzione
JasperServer	100	1,5 secondi	Seconda esecuzione
PoC	100	9,5 secondi	Prima esecuzione
PoC	100	1,6 secondi	Seconda esecuzione
JasperServer	1000	9,65 secondi	Prima esecuzione
JasperServer	1000	2,46 secondi	Seconda esecuzione

Soluzione	Dataset	Tempo di esecuzione	Note
PoC	1000	7,6 secondi	Prima esecuzione
PoC	1000	5,67 secondi	Seconda esecuzione

6.2. Accortezze per ottenere prestazioni efficaci

A seguito dei test effettuati si è notato che a una prima esecuzione entrambi i prodotti, JasperServer e il PoC, presentano un'esecuzione più lenta mentre nelle esecuzioni successive hanno tempi di esecuzione simili. Questo è dovuto al fatto che la prima esecuzione comporta il caricamento di tutte le librerie necessarie per eseguire il render e il popolamento del report mentre nelle esecuzioni successive queste restano in memoria, per cui si consiglia di eseguire un report di prova, anche vuoto, all'inizio per eseguire questa fase di caricamento delle librerie per ridurre in modo significativo i tempi di esecuzione e successivamente eseguire il report più complesso.

7. Approfondimenti futuri

Come approfondimenti futuri si può provare a sviluppare un server che possa incorporare l'esecuzione del programma realizzato per generare report in modo automatizzato tramite chiamata API dedicata, simulando quello che avviene con JasperServer.

8. Conclusioni

A seguito di tutto lo studio fatto è emerso che non sono disponibili molte soluzioni alternative al prodotto JasperServer in quanto esso fornisce funzionalità specifiche utilizzate dall'azienda che altre soluzioni prevedono in parte o in maniera incompleta. La realizzazione del PoC tramite Birt Runtime dimostra che è possibile utilizzare questo prodotto per realizzare i report nel modo più simile a JasperServer ma è allo stesso tempo limitato nel requisito fondamentale di gestione dei sottoreport e della parte server.

Dai risultati del PoC emerge chiaramente che nessuna delle soluzioni open-source analizzate può sostituire JasperServer in maniera completa. Tuttavia, l'analisi ha permesso di evidenziare punti di forza, limiti e criticità di ciascuna piattaforma, fornendo elementi oggettivi per eventuali scelte future e possibili miglioramenti architetturali.

Si racconmanda quindi di valutare se accettare compromessi funzionali e investire in sviluppo interno basato su BIRT runtime oppure mantenere JasperServer con licensing commerciale.