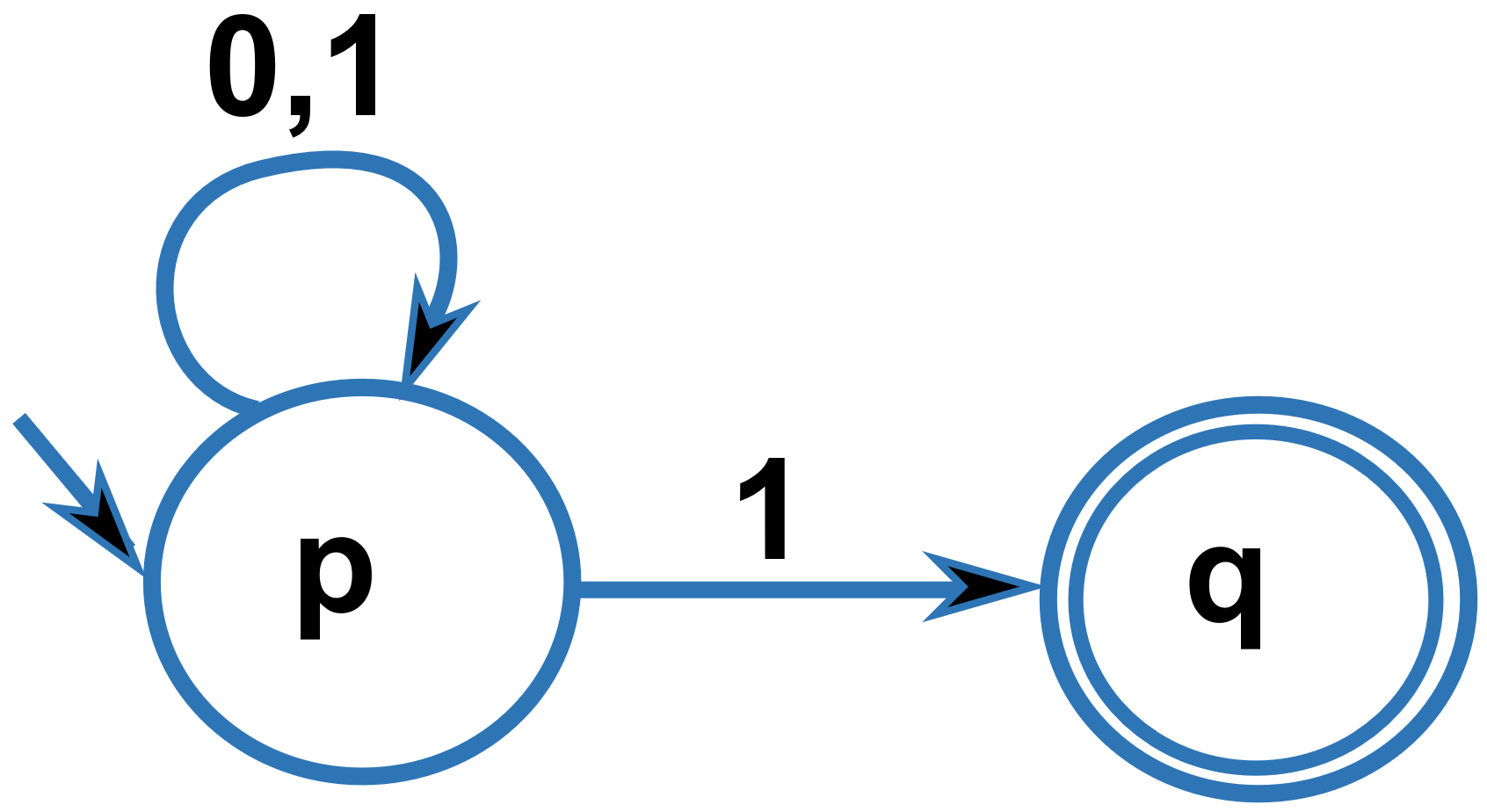
|  |
| --- |
| Mille Isole SRL |
| Flussu v2.0 |
| Uso & Programmazione |

|  |
| --- |
| Aldo Prinzi  25/09/2023 |



**Prefazione**

Questo manuale è dedicato a documentare Flussu, un avanzato server di processi sviluppato da Mille Isole SRL. Si tratta di una guida esaustiva che esplora varie funzionalità di Flussu, tra cui la definizione e l'esecuzione di processi, gli strumenti di editing grafico, la programmazione di blocchi di processo, il supporto multilingua, la registrazione dei dati, i comandi per l'esecuzione esterna, i sottoprocessi, i connettori API e i protocolli di comunicazione con sistemi esterni.

Flussu è progettato per offrire soluzioni agili a diversi problemi. Le sue funzionalità chiave includono:

* **Definizione ed Esecuzione dei Processi**

Flussu permette di definire ed eseguire processi in modo efficiente, supportando diversi livelli di esperienza di programmazione.

* **Strumenti di Editing Grafico**

Fornisce strumenti intuitivi per la progettazione e la modifica dei processi, accessibili sia a principianti che a esperti.

* **Blocchi di Programmazione**

Supporta la programmazione interna ai singoli blocchi di un processo, facilitando la gestione dei dati e l'interazione con eventi esterni e notifiche.

* **Supporto Multilingua**

Progettato per essere utilizzato in diverse lingue, sia nell'interfaccia utente che negli elementi del processo.

* **Registrazione dei Dati e Integrazione JSON**

Flussu può registrare dati durante l'esecuzione dei processi e supporta il formato JSON per facilitare l'integrazione con altri sistemi.

* **Comandi Esterni e Connettività API**

Questa guida dettaglia vari comandi esterni e opzioni di connettività API per migliorare l'interazione di Flussu con sistemi esterni.

* **Gestione Sottoprocessi**

Spiega come gestire e integrare i sottoprocessi all'interno di un flusso di lavoro principale.

* **Protocolli di Comunicazione**

Fornisce dettagli sui protocolli di comunicazione per l'interfaccia con sistemi esterni, assicurando scambi di dati sicuri ed efficienti.

* **Comandi Personalizzati e Testo Migliorato**:

Consente la creazione di comandi personalizzati e supporta testo migliorato per diversi elementi del processo.

* **Elementi dell'Interfaccia Utente**

Supporta vari elementi come pulsanti, etichette, caselle di testo ed elementi multimediali per interazioni dinamiche con gli utenti.

L'obiettivo principale di questo manuale è:

1. Facilitare l'implementazione di processi da parte di utenti con vari livelli di conoscenza tecnica.
2. Consentire a utenti e tecnici di collaborare nella realizzazione di soluzioni, progettando insieme il processo.
3. Agevolare la modifica e l'evoluzione dei processi.
4. Avvicinare i neofiti e i giovani al mondo della programmazione attraverso un approccio intuitivo e visuale.

Questa versione del manuale si concentra sulla versione 2.\* di Flussu.

|  |  |
| --- | --- |
|  |  |

*Questo manuale è stato scritto nel gennaio 2021, aggiornato a Settembre 2023.*

Sommario

[Definizione di processo 1](#_Toc150770957)

[**Il problem solving attraverso i processi** 2](#_Toc150770958)

[Cosa è un processo? 2](#_Toc150770959)

[Problemi e Ricerca di Soluzioni 2](#_Toc150770960)

[Schematizzazione a Blocchi 2](#_Toc150770961)

[Dalla Teoria alla Pratica 2](#_Toc150770962)

[Esempio di Processo 3](#_Toc150770963)

[Il Ciclo di Vita di una Soluzione di Processo 4](#_Toc150770964)

[L'Importanza dell'Evoluzione nei Processi: Adattabilità come Chiave per la Relevanza 4](#_Toc150770965)

[Adattabilità e Innovazione 4](#_Toc150770966)

[Risposta ai Cambiamenti 5](#_Toc150770967)

[Evoluzione o Abbandono 5](#_Toc150770968)

[Continuo Miglioramento 5](#_Toc150770969)

[Flussu: Uno Strumento per Processi Agili e Fluidi 6](#_Toc150770970)

[Applicazione Pratica della Teoria 6](#_Toc150770971)

[Innovazione nel Problem Solving 6](#_Toc150770972)

[Un Sistema Basato su Blocchi 6](#_Toc150770973)

[Interazione Collaborativa e Flessibilità 6](#_Toc150770974)

[Revisione e Ottimizzazione Continua 7](#_Toc150770975)

[Accessibilità e Intuitività 7](#_Toc150770976)

[Efficienza nella Modifica dei Processi 7](#_Toc150770977)

[WoFoBot: Un Motore Interno Agile: 7](#_Toc150770978)

[Strumenti di Flussu 7](#_Toc150770979)

[Definizione di WorkFlowBot 9](#_Toc150770980)

[WorkFlowBot – definzioni 10](#_Toc150770981)

[Block 10](#_Toc150770982)

[Exit 11](#_Toc150770983)

[Element 12](#_Toc150770984)

[Sistema multilingua: *var assign* 13](#_Toc150770985)

[Riepilogo 13](#_Toc150770986)

[Emoticons 14](#_Toc150770987)

[Interfaccia di configurazione dei processi 15](#_Toc150770988)

[Titolo 16](#_Toc150770989)

[Uso dell’editor per disegnare un processo 17](#_Toc150770990)

[Editing dei processi 18](#_Toc150770991)

[Funzionalità dell'Editor Grafico: 18](#_Toc150770992)

[Creazione e Gestione di Blocchi: 18](#_Toc150770993)

[Editing Visuale: 18](#_Toc150770994)

[Blocco di Start e Struttura del Workflow: 19](#_Toc150770995)

[Tipi di Blocchi: 19](#_Toc150770996)

[Gestione Multilingua: 19](#_Toc150770997)

[Funzionalità di Zoom e Navigazione: 19](#_Toc150770998)

[Visualizzazione Intuitiva di Elementi nei Blocchi: 20](#_Toc150770999)

[Blocchi di processo 21](#_Toc150771000)

[Programmazione dei blocchi 22](#_Toc150771001)

[Linguaggi di programmazione disponibili 22](#_Toc150771002)

[Interazione con server e servizi esterni via API/JSON 22](#_Toc150771003)

[Convenzioni di Scrittura in PHP 23](#_Toc150771004)

[Variabili 24](#_Toc150771005)

[Validità delle variabili 24](#_Toc150771006)

[Comandi predefiniti nel motore 25](#_Toc150771007)

[Environment 25](#_Toc150771008)

[init 25](#_Toc150771009)

[endscript 25](#_Toc150771010)

[Impostazioni della lingua 26](#_Toc150771011)

[getLang 26](#_Toc150771012)

[setLang 26](#_Toc150771013)

[Selezione delle uscite 27](#_Toc150771014)

[setExit 27](#_Toc150771015)

[getExit 27](#_Toc150771016)

[Indicatori di stato e di fonte della richiesta di esecuzione 28](#_Toc150771017)

[Identificazione della Fonte della Chiamata: 28](#_Toc150771018)

[Personalizzazione dell'Interazione in Base alla Piattaforma: 28](#_Toc150771019)

[In Caso di Typeform: 28](#_Toc150771020)

[In Caso di Chat: 29](#_Toc150771021)

[Generazione della Data e Formato: 30](#_Toc150771022)

[Generazione di dati casuali 31](#_Toc150771023)

[getRnd 31](#_Toc150771024)

[generateNewCheckCode 31](#_Toc150771025)

[generateNewPassword 31](#_Toc150771026)

[Registratore dei dati 33](#_Toc150771027)

[recData 33](#_Toc150771028)

[getData 33](#_Toc150771029)

[getDataJson 33](#_Toc150771030)

[setDataJson 33](#_Toc150771031)

[Estrattore JSON di array di dati 34](#_Toc150771032)

[makeJson 34](#_Toc150771033)

[Esecuzioni esterne 35](#_Toc150771034)

[sendEmail 35](#_Toc150771035)

[sendPremiumEmail 35](#_Toc150771036)

[print2PDF (v2.9) 36](#_Toc150771037)

[sendEmail con fileattach (v2.9) 36](#_Toc150771038)

[Integrazione ZAPIER 37](#_Toc150771039)

[Integrazione OpenAi 38](#_Toc150771040)

[Utilità 39](#_Toc150771041)

[checkEmailAddress 39](#_Toc150771042)

[isDisposableEmailAddress 39](#_Toc150771043)

[generateQrCode 39](#_Toc150771044)

[checkCodFiscale 39](#_Toc150771045)

[Flussu Enhanced Text e Gestione delle risposte 41](#_Toc150771046)

[Tag per la Modifica del Testo 41](#_Toc150771047)

[Tag per la Gestione della Pagina 41](#_Toc150771048)

[Tag per Elementi Multimediali 42](#_Toc150771049)

[Sub-Processi (v2.0) 43](#_Toc150771050)

[Gestione dei sottoprocessi 44](#_Toc150771051)

[Inizializzazione e Variabili nei Sottoprocessi 44](#_Toc150771052)

[Integrazione delle Variabili e dello stato 44](#_Toc150771053)

[Gestione dei Blocchi Terminali in Sottoprocessi 44](#_Toc150771054)

[Processi multipli (v3.0) 45](#_Toc150771055)

[Gestione dei processi multipli 46](#_Toc150771056)

[Integrazioni Service Oriented (SOA) 47](#_Toc150771057)

[Protocollo di colloquio con sistemi esterni 48](#_Toc150771058)

[getXCmdKey 49](#_Toc150771059)

[sendXCmdData 49](#_Toc150771060)

[Implementazione del protocollo di comunicazione 50](#_Toc150771061)

[Autenticazione/autorizzazione 50](#_Toc150771062)

[Esecuzione del comando remoto 51](#_Toc150771063)

[Connettore API / Esecuzione chatbot 54](#_Toc150771064)

[Protocollo di esecuzione 55](#_Toc150771065)

[Modalità informativa 55](#_Toc150771066)

[Modalità esecutiva 56](#_Toc150771067)

[Startup dallo start block 56](#_Toc150771068)

[Ad esempio 57](#_Toc150771069)

[Termini 58](#_Toc150771070)

[Restituzione parametri 58](#_Toc150771071)

[Startup da un blocco arbitrario 59](#_Toc150771072)

[Modalità assegnazione valori arbitrari ai parametri 59](#_Toc150771073)

[Passaggio di valori di startup/arbitrari 60](#_Toc150771074)

[Esecuzione chatbot da CHAT APP standard 61](#_Toc150771075)

[Esecuzione da chat app standard 62](#_Toc150771076)

[Introduzione ai Bot di Telegram e l'Integrazione con Flussu 62](#_Toc150771077)

[Cosa Sono i Bot di Telegram 62](#_Toc150771078)

[Integrazione di Flussu con Telegram 62](#_Toc150771079)

[Come Funziona nella Pratica 62](#_Toc150771080)

[Vantaggi dell'Integrazione di Flussu con Telegram 63](#_Toc150771081)

[Bot Father 63](#_Toc150771082)

[Introduzione alle chat WhatsApp e l'Integrazione con Flussu 65](#_Toc150771083)

[Necessità di un Numero di Telefono 65](#_Toc150771084)

[Configurazione Tramite META 65](#_Toc150771085)

[Passaggi della Configurazione 65](#_Toc150771086)

[Sfide e Considerazioni 66](#_Toc150771087)

[Vantaggi dell'Integrazione con WhatsApp 66](#_Toc150771088)

[Dati statistici 67](#_Toc150771089)

[Dati statistici 68](#_Toc150771090)

[Integrazione 68](#_Toc150771091)

[Limiti 68](#_Toc150771092)

[Sessioni d’uso 69](#_Toc150771093)

[Sessioni d’uso 70](#_Toc150771094)

[Integrazione 70](#_Toc150771095)

[Limiti 70](#_Toc150771096)

[Flussu APP Android+iOs (v3.0) 71](#_Toc150771097)

[Flussu App 72](#_Toc150771098)

[Integrazione 72](#_Toc150771099)

[Limiti 72](#_Toc150771100)

[**Programmazione avanzata**: 73](#_Toc150771101)

[motore server di Flussu 73](#_Toc150771102)

[Classe Environment 75](#_Toc150771103)

[Elementi del server 75](#_Toc150771104)

[Classi 75](#_Toc150771105)

[Flussut\Flussut 76](#_Toc150771106)

[Flussut\WofoCmd 76](#_Toc150771107)

[Flussut\WofoEnv 77](#_Toc150771108)

[Flussut\WofoSess 77](#_Toc150771109)

[Flussut\WofoWork 77](#_Toc150771110)

[Editing e rappresentazione dei processi 78](#_Toc150771111)

[Forma dei dati 79](#_Toc150771112)

[Metodo d’uso 79](#_Toc150771113)

[Comandi disponibili 79](#_Toc150771114)

[GET Workflow List 79](#_Toc150771115)

[Create NEW workflow 80](#_Toc150771116)

[GET Workflow 80](#_Toc150771117)

[Editing 80](#_Toc150771118)

[Update workflow 80](#_Toc150771119)

[GET block 81](#_Toc150771120)

[Editing 81](#_Toc150771121)

[Update Block 81](#_Toc150771122)

[Delete Block 82](#_Toc150771123)

[Duplicate Block 83](#_Toc150771124)

[Trasferimento Workflow 84](#_Toc150771125)

[Estrazione schema di un workflow d’esempio 85](#_Toc150771126)

[Database di servizio a Flussu 87](#_Toc150771127)

[Database 88](#_Toc150771128)

[Disegno logico delle tabelle del processo 88](#_Toc150771129)

[Processo 89](#_Toc150771130)

[Backup copy 89](#_Toc150771131)

[Blocco 90](#_Toc150771132)

[Uscite dei blocchi 91](#_Toc150771133)

[Elementi 92](#_Toc150771134)

[Testo in lingua 93](#_Toc150771135)

[Disegno logico delle tabelle degli utenti 94](#_Toc150771136)

[Utenti del sistema 94](#_Toc150771137)

[Ruoli degli utenti 95](#_Toc150771138)

[Esecuzione dei processi 96](#_Toc150771139)

[Dati della sessione del processo 96](#_Toc150771140)

[Dati delle variabili della sessione 97](#_Toc150771141)

[Dati del log della sessione 97](#_Toc150771142)

[Dati statistici 98](#_Toc150771143)

[Dati delle sessioni d’uso 98](#_Toc150771144)

[Canali d’accesso 99](#_Toc150771145)

# Definizione di processo

## Il problem solving attraverso i processi

# Cosa è un processo?

La risoluzione dei problemi attraverso i processi si basa su un approccio strutturato e organizzato per affrontare e risolvere le sfide.

Un "processo" è un insieme di attività o passaggi ordinati, diretti alla risoluzione di un problema o al raggiungimento di un obiettivo specifico.

Questa metodologia è efficace sia nelle questioni quotidiane sia in contesti più complessi.

## Problemi e Ricerca di Soluzioni

In ogni situazione problematica, il primo passo fondamentale è l'identificazione del problema.

Una volta riconosciuto, il problema deve essere definito in modo chiaro e preciso per permettere una ricerca efficace di soluzioni potenziali.

## Schematizzazione a Blocchi

La schematizzazione a blocchi è una tecnica di problem solving che prevede la rappresentazione grafica del problema e delle sue soluzioni attraverso una serie di passaggi sequenziali.

Ogni "blocco" rappresenta un'azione specifica o una decisione chiave nel processo. Questo metodo di rappresentazione visuale ha il vantaggio di essere immediatamente comprensibile, anche per chi non è direttamente coinvolto nel problema, rendendolo un efficace strumento di comunicazione per processi complessi.

## Dalla Teoria alla Pratica

Nella pratica, trasformare un problema astratto in una serie di azioni concrete attraverso la schematizzazione a blocchi aiuta a chiarire il percorso verso la soluzione. Ad esempio, consideriamo una situazione semplice come "Ho fame":

Problema: Ho fame.

Soluzione:

* Prendere i soldi.
* Chiudere la porta.
* Andare a comprare il pane.
* Tornare a casa.
* Preparare un panino.
* Mangiare il panino.

In questo esempio, ogni passo necessario per risolvere il problema ("Ho fame") è chiaramente delineato in una sequenza logica di azioni.

Ciò non solo facilita la comprensione del processo necessario per risolvere il problema, ma fornisce anche una struttura chiara e ordinata per l'attuazione della soluzione.

Questo approccio è altamente efficace per una vasta gamma di problemi, consentendo di suddividere compiti complessi in passaggi gestibili e facilmente comprensibili.

## Esempio di Processo

Quando iniziamo a schematizzare un processo, la nostra prima rappresentazione tende a focalizzarsi sugli aspetti più evidenti e diretti del problema.

|  |
| --- |
|  |
| *schematizzazione “a blocchi” di un processo* |

Prendiamo, ad esempio, il problema legato all'acquisto di un computer portatile, quale sarà il “processo minimo” di cui abbiamo bisogno per risolvere il problema? Ci facciamo aiutare dalla semplicità della schematizzazione dei diagrammi di flusso[[1]](#footnote-1):

Inizialmente, il processo può essere rappresentato con passaggi semplici come selezionare un modello, trovare un rivenditore e effettuare l'acquisto.

Tuttavia, questa rappresentazione iniziale spesso omette molti dettagli cruciali che sono fondamentali per la realizzazione effettiva del processo.

Ciò che inizia come un processo apparentemente semplice, si evolve sempre rapidamente in un processo più complesso con molteplici decisioni e cicli.

Il diagramma nella figura, sebbene sia una buona schematizzazione nei fatti è solo un punto di partenza “minimo”.

|  |
| --- |
|  |
| *Evoluzione del processo precedente* |

## Il Ciclo di Vita di una Soluzione di Processo

### L'Importanza dell'Evoluzione nei Processi: Adattabilità come Chiave per la Relevanza

La capacità di un processo di evolvere è fondamentale per la sua applicabilità e accettabilità a lungo termine.

In un mondo in continuo cambiamento, un processo che rimane statico è destinato a diventare obsoleto e, di conseguenza, viene spesso abbandonato.

L'evoluzione costante è quindi non solo desiderabile ma essenziale per la sopravvivenza e l'efficacia di un processo.

### Adattabilità e Innovazione

I processi devono essere flessibili per adattarsi a nuove scoperte, innovazioni e cambiamenti nel contesto in cui operano.

Questa adattabilità permette ai processi di restare aggiornati con le ultime tendenze, tecnologie e metodologie, assicurando che siano sempre al passo con i tempi e rispondano efficacemente alle nuove esigenze.

### Risposta ai Cambiamenti

In un ambiente dinamico, i processi devono essere in grado di rispondere rapidamente a cambiamenti esterni, come evoluzioni del mercato, nuove normative o progressi tecnologici.

Un processo che non può adattarsi a tali cambiamenti rischia di diventare inutile o controproducente, portando le organizzazioni a cercare alternative più flessibili e responsive.

### Evoluzione o Abbandono

Un processo che non evolve diventa un ostacolo anziché un vantaggio. S

enza la capacità di adeguarsi e migliorare, i processi statici sono destinati ad essere abbandonati in favore di approcci più agili e adattabili.

|  |
| --- |
|  |
| *Ciclo di vita di una soluzione sotto forma di processo* |

Questo abbandono non è solo una perdita di risorse, ma anche un'opportunità mancata per crescere e innovare.

### Continuo Miglioramento

L'obiettivo di ogni processo dovrebbe essere quello di perseguire un miglioramento continuo.

Questo non significa solo apportare piccoli aggiustamenti nel tempo, ma anche essere pronti a riconsiderare e ristrutturare il processo quando necessario per assicurare che rimanga efficace e pertinente.

In conclusione, la capacità di evoluzione è un attributo critico per qualsiasi processo. In un'epoca caratterizzata da rapidi cambiamenti e innovazioni continue, solo i processi che possono adattarsi e evolvere saranno in grado di sopravvivere e prosperare.

L'evoluzione non è solo un segno di flessibilità; è un requisito indispensabile per l'efficacia e la sostenibilità a lungo termine dei processi.

## Flussu: Uno Strumento per Processi Agili e Fluidi

Basandoci sul concetto che i processi necessitano di evoluzione continua per rimanere efficaci, abbiamo sviluppato Flussu, un acronimo di *Flusso di Processo*.

**Flussu** si propone come uno strumento rivoluzionario, che incarna oltre tre decenni di esperienza nel settore dello sviluppo software, per gestire, facilitare e ottimizzare la creazione e l'esecuzione di processi.

### Applicazione Pratica della Teoria

In Flussu, la teoria dell'evoluzione dei processi trova una sua applicazione pratica. Il processo si trasforma in un flusso di lavoro strutturato, dove ogni blocco rappresenta un componente essenziale di un puzzle più ampio.

Questo approccio modulare consente di visualizzare e gestire facilmente ogni fase del processo.

### Innovazione nel Problem Solving

Flussu rappresenta un cambiamento radicale rispetto ai metodi tradizionali di risoluzione dei problemi e sviluppo di software.

Al centro della sua ideazione c'è la consapevolezza che i problemi e le loro soluzioni evolvono nel tempo, rendendo necessaria una continua revisione e adattamento dei processi.

### Un Sistema Basato su Blocchi

A differenza dei programmi tradizionali, che sono statici e autocontenuti, Flussu adotta un'architettura basata su blocchi in sequenza.

Questa struttura rende il processo facilmente comprensibile e adattabile, semplificando significativamente sia la prototipazione sia le fasi successive di evoluzione e modifica.

### Interazione Collaborativa e Flessibilità

Flussu non limita la gestione e la modifica dei processi al loro creatore originale. Questa caratteristica lo rende particolarmente prezioso in ambienti collaborativi o in situazioni in cui il creatore non è disponibile, poiché terzi possono facilmente intervenire per aggiornare o modificare i processi.

### Revisione e Ottimizzazione Continua

La piattaforma supporta l'adattamento agile a nuovi cambiamenti, eccezioni impreviste e l'introduzione di nuove esigenze.

La sua architettura a blocchi facilita la revisione e l'ottimizzazione continua dei processi.

### Accessibilità e Intuitività

La natura intuitiva e auto-documentante dei sistemi basati sulla schematizzazione a blocchi rende ogni processo trasparente e accessibile, anche a coloro che non hanno partecipato alle fasi iniziali di sviluppo.

### Efficienza nella Modifica dei Processi

Grazie alla sua architettura modulare, Flussu consente modifiche rapide ed efficienti, senza la necessità di riscrivere l'intero processo. Questa flessibilità lo rende ideale per ambienti dinamici e in costante evoluzione.

## WoFoBot: Un Motore Interno Agile:

Il motore interno di Flussu, denominato WoFoBot (WOrkFlOw-roBot), è progettato per rendere i processi più agili e gestibili.

La sua capacità di facilitare modifiche e revisioni in risposta ai cambiamenti lo rende uno strumento essenziale non solo per la creazione di soluzioni, ma anche per la loro evoluzione e modifica nel tempo, semplificando l'intero ciclo di vita del processo di problem solving.

In conclusione, Flussu si posiziona come un metodo innovativo per la soluzione dei problemi e come un alleato fondamentale nella gestione dinamica, nell'adattamento e nell'evoluzione dei processi, enfatizzando l'importanza di un approccio flessibile e adattabile nel mondo del problem solving.

## Strumenti di Flussu

Flussu aiuta a rendere la manutenzione dei processi semplice e agile attraverso:

Editing Grafico

Uno strumento di editing grafico permette a chiunque, dal principiante all'esperto, di delineare un processo in modo visivo e intuitivo. Il disegno viene poi trasformato in istruzioni eseguibili dal motore di processo.

Programmabilità degli Elementi

Ogni passo del processo (da ora in poi definito *blocco* di Flussu) può contenere codice per eseguire comandi specifici, consentendo di costruire database, gestire stati, prendere decisioni, reagire a eventi e inviare notifiche.

Interfaccia Utente

I blocchi di Flussu possono includere elementi per interfacce utente, come schermate per raccogliere informazioni o fornire feedback. Un'interfaccia disponibile simula una chat con un utente remoto, facilitando l'interazione in modo naturale.

Integrato nel concetto di SOA (Service-Oriented Architecture)

La filosofia di sviluppo di soluzioni software basata sull'architettura a microservizi si è sviluppata e diffusa principalmente nel corso del terzo millennio e ha iniziato a guadagnare popolarità intorno al 2010.

L'architettura a microservizi trae le sue origini dall'architettura orientata ai servizi (SOA), un modello di design software che prevede la creazione di applicazioni composte da servizi interconnessi per necessità di agilità e scalabilità.

Con l'evolversi delle pratiche di sviluppo software e l'aumento delle esigenze di business, le applicazioni monolitiche tradizionali, in cui l'intero software è un unico blocco indivisibile, hanno iniziato a mostrare limiti in termini di flessibilità e gestione della complessità.

La soluzione è utilizzare una architettura a microservizi, che risponde alle esigenze di maggiore agilità, scalabilità e velocità nell'evoluzione del software, in un contesto in cui le tecnologie cloud e le pratiche di sviluppo agile stavano diventando sempre più predominanti.

Flussu è stato realizzato avendo in mente questo tipo di architettura. Il server Flussu è a tutti gli effetti un elemento di una architettura SOA, utilizzato attraverso API/JSON e naturalmente in grado di comunicare con altri sistemi nello stesso modo.

# Definizione di WorkFlowBot

# WorkFlowBot – definzioni

**Flussu** si distingue come uno strumento versatile per la definizione, modifica ed esecuzione di processi.

Tecnicamente, Flussu funziona come un sistema di "automi a stati finiti" non deterministici, ideale per gestire processi di varia complessità attraverso un calcolatore.

La forza di Flussu risiede in una interfaccia grafica intuitiva, che consente una progettazione agevole e diretta dei processi e questo strumento grafico trasforma la complessità di gestione dei processi in un'attività semplice e visivamente comprensibile, rendendo la realizzazione di processi complessi molto più accessibile.

Ogni processo in Flussu è strutturato in una serie di blocchi, ciascuno eseguito in modo sequenziale.

Il flusso inizia da un blocco di "start", che segna il punto di partenza del processo e da lì in poi, il processo si sviluppa attraverso una serie di blocchi che vengono eseguiti uno dopo l'altro, in base allo stato corrente del processo.

Questo percorso continua fino a quando non si raggiunge un blocco che non presenta ulteriori uscite, segnalando così il termine del processo.

La rappresentazione grafica di Flussu è stata ideata per offrire una visione chiara e organizzata di ogni passaggio del processo, facilitando non solo la comprensione ma anche l'identificazione di aree che potrebbero richiedere modifiche o miglioramenti.

Attraverso questa visualizzazione, Flussu trasforma l'astrazione dei processi in una realtà concreta e gestibile, rendendo l'intero sistema di gestione dei processi efficiente e facilmente navigabile.

Analizziamo da qui in poi i “concetti” introdotti nel sistema di progettazione dei processi:

## Block

Ogni blocco possiede un codice identificativo unico (**BID** – *BlockID*) e si compone di:

* uno strato eseguibile;
* da zero a N elementi;
* almeno due uscite.

|  |
| --- |
| Blocco ***1223-F5A32B-175A25***  var init=1;  var exit=0;  json\_encode(array{“flussu   * LABEL * TEXTBOX * BUTTON * IMAGE * VIDEO * LANG/TEXT\_ASSIGN   Uscita 0 .  Uscita ... N |
| *Fig.A – Anatomia di un blocco* |

Lo strato **eseguibile** è il fulcro di questo manuale e sarà dettagliato nei capitoli successivi.

Gli **elementi** rappresentano la parte visibile del blocco, come etichette, caselle di testo, pulsanti, immagini, video e, come vedremo più avanti nella sezione "multilinguaggio", anche un sistema per l'assegnazione di testo in diverse lingue.

Le **uscite**, numerate da 0 a N, sono i punti di transizione verso altri blocchi. Il processo procede eseguendo un blocco alla volta, muovendosi verso il blocco successivo attraverso una delle uscite. La scelta di quale uscita utilizzare dipende dal codice eseguibile o da elementi interattivi come i *button*.

Un **processo** inizia con l'esecuzione del *blocco di start*, la cui *uscita 0* conduce al primo *blocco* definito dall'utente, proseguendo poi attraverso i blocchi collegati, e termina quando un blocco eseguito ha entrambe le uscite che *non portano a nessun altro blocco*.

## Exit

Il motore di *workflow*, durante l'esecuzione di un *blocco*, decide quale *uscita* scegliere per proseguire il *processo*.

Questa scelta può essere determinata da un comando di programmazione specifico (vedi wofoDev->setExit) o richiesta all’utente attraverso elementi interattivi come pulsanti.

**Se nessuna scelta è stata fatta** e l'uscita 0 conduce a un altro blocco, il motore la selezionerà per default.

|  |
| --- |
| Blocco **B**  Blocco **A**  Uscita 0  Uscita 1  Blocco **C** |
| *Fig.B - Collegamento tra blocchi* |

## Element

L’esecuzione di un *processo* implica l'attivazione di *blocchi* o passaggi che modificano lo *stato del processo* in corso, a seconda degli elementi rilevabili dal sistema che lo esegue.

Ad esempio, un processo che identifica una email e la sposta in un altro contenitore prima di stamparla, può operare autonomamente in background senza interazione umana.

Tuttavia, ci sono processi che richiedono l'interazione con gli utenti per cambiare stato, memorizzare dati o prendere decisioni.

In questi casi, Flussu prevede l'uso di una *maschera di input*; ogni blocco può includere vari elementi progettati per interagire con l'utente:

* **Label**: testo da visualizzare;
* **Input**: casella di richiesta di inserimento di un dato;
* **Button**: pulsante;
* **Media**: foto o video;
* **Link**: collegamento ipertestuale;
* **Var assign**: assegnazione testo (*cfr:* paragrafo successivo);
* **Select**: lista selezionabile.
* **Upload media**: acquisizione foto, documenti, ecc.

**Flussu**, essendo multilingua, permette che ogni elemento abbia impostazioni diverse a seconda della lingua scelta dall'utente.

La selezione di foto, video o link può variare per ogni lingua, considerando che le immagini possono contenere testo in diverse lingue, i video potrebbero essere doppiati, e i link potrebbero condurre a pagine web in lingue specifiche.

## Sistema multilingua: *var assign*

Gli *element* sono gli unici componenti del processo che consentono di impostare valori differenti a seconda della lingua scelta dall'utente.

L'obiettivo degli *element* è creare uno schema di visualizzazione. Se il processo, ad esempio, compone un testo per una email, gran parte di esso sarà predefinito durante la progettazione.

Con Flussu, è possibile definire testi diversi per email in lingue diverse.

L'element "***var assign***" serve proprio a impostare una variabile con un testo fisso differenziato per lingua, senza necessità di visualizzarlo nella finestra di interazione.

## Riepilogo

Per illustrare meglio i concetti descritti, consideriamo un processo per raccogliere e registrare l'apprezzamento di un utente verso un prodotto:

|  |
| --- |
| Blocco **2**  Blocco **1**    **SI**  Init block  Blocco **4**  var init=1;  var exit=0;  json\_encode(array{“wofobo  Ti piace?  Grazie 😊  Blocco **3**  **NO**  **SI**  **NO** |
| *Fig.C – Processo d’esempio* |

Nella Fig. C vediamo lo **start block**, il punto di inizio di ogni processo in *Flussu*.

Questo blocco inizializza il processo e non contiene elementi interattivi. Come gli altri blocchi, ha due uscite: 0 e 1.

Se non viene selezionata un'uscita specifica, verrà utilizzata l'*uscita 0* per default.

Il motore di workflow, dopo aver eseguito il codice di questo blocco, procederà con il *blocco* 1, che contiene quattro element: un'immagine del prodotto, una etichetta con la domanda "ti piace?", e due pulsanti per le risposte "SÌ" e "NO".

* **media**: l’immagine del prodotto
* **label**: “ti piace?”
* **pulsante** 1: SI
* **pulsante** 2: NO

La scelta dell'utente determina il blocco successivo: l'uscita 0 conduce al blocco 2, l'uscita 1 al blocco 3.

Il blocco 1 può presentare elementi in diverse lingue, mentre la registrazione dello stato nel processo nei blocchi 2 e 3 è indipendente dalla lingua scelta.

Sia il blocco 2 che il 3, una volta registrato lo stato, indirizzano al blocco 4 attraverso l'uscita 0.

Quest'ultimo contiene un'etichetta di ringraziamento, nella stessa lingua scelta all'inizio del processo.

Poiché non è collegato ad altri blocchi, il processo si considera concluso con il blocco 4.

## Emoticons

Nel costante impegno di rendere l'interfaccia utente di Flussu non solo funzionale ma anche più accattivante e piacevole, abbiamo introdotto la possibilità di arricchire il testo delle etichette (label) con emoticons sotto forma di emoji[[2]](#footnote-2).

Questo piccolo ma significativo dettaglio aggiunge un tocco di simpatia e immediatezza all'interfaccia, migliorando l'esperienza utente.

# Interfaccia di configurazione dei processi

L’interfaccia di configurazione del processo è descritta in questo capitolo

# Titolo

Lorem Ipsum **TBD**

# Uso dell’editor per disegnare un processo

# Editing dei processi

L'editor grafico di Flussu è uno strumento avanzato e intuitivo che opera all'interno del browser, progettato per facilitare la creazione e la gestione di processi complessi attraverso una visualizzazione chiara e interattiva.

Per una performance ottimale, l'editor è raccomandato per l'uso con browser come Chrome e Microsoft Edge, basati su Chromium, che supportano funzionalità avanzate come lo zoom su oggetti specifici.

Alla data della realizzazione di questo manuale, browser come Firefox non supporta tutte le modalità offerte dall'editor.

## Funzionalità dell'Editor Grafico:

Compatibilità con il Browser: Ottimizzato per Chrome e Microsoft Edge per sfruttare al meglio le loro capacità di gestione grafica avanzata.

## Creazione e Gestione di Blocchi:

L'editor permette di aggiungere blocchi, definirne le proprietà, gli elementi, integrare codice PHP, gestire le uscite e i connettori tra i vari blocchi.

|  |  |
| --- | --- |
|  |  |

## Editing Visuale:

L'interfaccia puramente visuale consente di aggiungere o cancellare blocchi con facilità. La manipolazione visiva degli elementi rende l'editor estremamente user-friendly.

|  |
| --- |
|  |

## Blocco di Start e Struttura del Workflow:

Ogni workflow inizia con un blocco di start che definisce il punto di partenza del processo. Questo blocco fondamentale non può essere rimosso, garantendo la coerenza strutturale del workflow.

## Tipi di Blocchi:

L'editor offre diversi tipi di blocchi, tra cui quelli standard, blocchi per richiamare sottoprocessi, blocchi per terminare sottoprocessi e ritornare al processo principale, e blocchi per annotazioni testuali.

|  |
| --- |
|  |

## Gestione Multilingua:

I processi iniziano con una definizione linguistica e possono essere tradotti in altre lingue.

L'editor mostra la lingua selezionata dall'utente, con la possibilità di modificare le etichette nei singoli elementi di ciascun blocco.

|  |
| --- |
|  |

## Funzionalità di Zoom e Navigazione:

Per gestire processi complessi e articolati, l'editor offre funzioni di zoom e spostamento del foglio di lavoro, facilitando la visualizzazione e l'editing di workflow con numerosi blocchi.

|  |
| --- |
|  |

## Visualizzazione Intuitiva di Elementi nei Blocchi:

Ogni blocco mostra visivamente se contiene codice, elementi visivi o immagini, rendendo immediatamente chiaro il contenuto e la funzione di ciascun blocco agli utenti.

|  |
| --- |
|  |

L'editor grafico di Flussu è dunque uno strumento potente e versatile, ideale per la realizzazione di processi sofisticati e per la loro gestione nel tempo. La sua interfaccia intuitiva e le funzionalità avanzate lo rendono adatto sia per utenti esperti sia per principianti nel campo della gestione dei processi.

# Blocchi di processo

# Programmazione dei blocchi

Ogni *blocco* in Flussu è programmabile, consentendo, ad esempio, la memorizzazione di valori in variabili, la selezione delle uscite, la generazione di codici casuali, e altre funzionalità.

Il codice programmato viene eseguito prima della visualizzazione degli elementi nel blocco.

Se il codice prevede una elaborazione “esterna” e i risultati influenzano gli elementi da visualizzare, diventa necessario suddividere il processo in due blocchi separati: uno per l'esecuzione del codice e l'altro per la visualizzazione degli elementi.

## Linguaggi di programmazione disponibili

Dalla versione 1.0 di Flussu, il linguaggio di programmazione adottato è il **PHP** perché è un linguaggio di programmazione di scripting (non compilato, interpretato ed eseguito) come il JavaScript, ma più usato come linguaggio di programmazione back-end, nel momento in cui Flussu è stato realizzato.

Le versioni comprendono sia la v7.4.x che la v8.1.x.

Sebbene non è stato considerato nel processo di evoluzione alla v3.0, nelle versioni future potrebbero essere integrati altri linguaggi di scripting come ad esempio JavaScript (ecmascript >9).

È possibile utilizzare i comandi base di PHP, tuttavia, alcune funzioni come include, eval, file, ecc., sono escluse per motivi di sicurezza, dato che l'esecuzione avviene all'interno di una "sandbox" che impedisce l'esecuzione di comandi esterni o potenzialmente dannosi per il server che esegue Flussu.

L'interazione con sistemi esterni, come l'invio di email, avviene tramite il motore di Flussu, che gestisce la raccolta e l'elaborazione di dati e comandi. Le informazioni relative a quale server di posta usare, all’autenticazione, protocollo ecc. è definito in un file di configurazione e spiegata in un paragrafo apposito.

Lo stesso avviene per l’interazione con OpenAI, server per l’invio di SMS, ecc.

La personalizzazione spinta di queste interazioni è facilitata da una classe PHP dedicata, situata in un file esterno, che può essere estesa e modificata secondo le più specifiche necessità.

## Interazione con server e servizi esterni via API/JSON

Flussu è un server API che interagisce attraverso chiamate http e scambio dei dati utilizzando il formato di rappresentazione JSON.

Così come utilizza server email, sms e OpenAi, Flussu può collegarsi a server esterni e scambiare dati, seguendo la filosofia dei microservizi.

Poiché fanno parte di un unico blocco di script eseguito all'inizio del blocco funzionale, anche i comandi di interazione con server esterni vengono processati in questa fase.

Pertanto, il contenuto delle email da inviare, degli SMS da trasmettere o della interazione tra interrogazione e risposta di OpenAI, deve essere preparata nel blocco precedente, mentre l'analisi dei risultati ottenuti da questi sistemi esterni deve avvenire in un blocco successivo.

## Convenzioni di Scrittura in PHP

Nella programmazione PHP in Flussu, ogni riga termina con un punto e virgola ";", seguendo la convenzione standard del linguaggio PHP.

La sandbox di Flussu rispetta questa normativa, garantendo coerenza e prevedibilità nello sviluppo del codice.

# Variabili

In Flussu sono supportate tutte le variabili previste dalla versione PHP v7.4.x[[3]](#footnote-3).

Come in PHP, la variabile è preceduta dal simbolo "$", che ne indica l'inizio.

## Validità delle variabili

Le variabili definite in Flussu, così come accade per i linguaggi di scripting, mantengono la loro validità a partire dal blocco in cui sono state create e nei blocchi successivi, poiché PHP consente l'utilizzo di variabili anche senza una dichiarazione esplicita.

Di conseguenza, si potrebbe incorrere nella situazione in cui si crede di validare il contenuto di una variabile, che però, non essendo stata precedentemente definita, viene generata automaticamente con un valore vuoto al momento del primo utilizzo.

Pertanto, se una variabile è necessaria in più blocchi e il suo utilizzo è critico, *si raccomanda* di dichiararla nel blocco di start.

Ciò garantisce che la variabile sia disponibile e inizializzata in tutti i blocchi del processo.

# Comandi predefiniti nel motore

L'ambiente di esecuzione in *sandbox*[[4]](#footnote-4) di Flussu impone alcune restrizioni sull'uso del linguaggio PHP, ma i comandi predefiniti incorporati nel motore di Flussu coprono efficacemente la maggior parte delle esigenze.

Nonostante non sia possibile anticipare ogni possibile necessità, i programmatori con esperienza in PHP hanno la possibilità di estendere l'insieme di comandi base, come verrà illustrato in un capitolo dedicato più avanti nel manuale.

## Environment

Il codice creato dagli utenti in Flussu viene eseguito all'interno di una sandbox per garantire sicurezza e controllo. In questo contesto, tutti i comandi sono eseguiti tramite un oggetto denominato **environment**, utilizzato nel seguente modo:

wofoEnv -> comando (parametri);

## init

Il comando per l'inizializzazione di un processo in Flussu è init. Questo comando è eseguito nel blocco di avvio del processo:

wofoEnv->init();

## endscript

Il comando endscript segna la conclusione del processo. La posizione esatta in cui il processo termina è definita dall'utente, quindi il comando endscript viene inserito manualmente dall'utente nell'ultimo blocco del processo:

wofoEnv->endScript();

# Impostazioni della lingua

La lingua predefinita per un processo è stabilita nelle sue proprietà.

Durante l'avvio del processo, è possibile passare un parametro specifico per indicare al motore Flussu quale, delle lingue definite, si deve utilizzare nello specifico momento di esecuzione del processo.

Durante l'esecuzione, tutte le interazioni testuali vengono gestite automaticamente dal motore, che seleziona una delle lingue definite.

Gli script all'interno di Flussu permettono di riconoscere la lingua attualmente selezionata o, se necessario, di cambiarla.

## getLang

Come suggerisce il nome del comando, attraverso esso si ottiene il codice della lingua selezionata (IT, EN, FR, DE, ecc.);

Restituisce una stringa.

$foo = wofoEnv->getLang();

## setLang

Attraverso questo comando è possibile indicare al motore di processo, di usare una determinata lingua (IT, EN, FR, DE, ecc.)

Richiede un parametro stringa.

wofoEnv->setLang(“EN”);

# Selezione delle uscite

Un blocco, come detto, può avere due o più uscite.

Ogni uscita rappresenta un percorso del motore di processo e quella prescelta può essere selezionata in base a un determinato stato.

Un blocco contiene almeno 2 uscite e ne può contenere fino a 64.

Se nessuna uscita è stata selezionata, il motore verifica se l’uscita 0 è collegata ad un altro blocco e, se si, esegue il blocco successivo.

## setExit

Nel caso in cui, ad esempio, si intenda selezionare una uscita differente, sarà possibile usare il comando setExit.

Richiede un parametro integer pari al numero dell’uscita che si vuole selezionare.

wofoEnv->setExit(1);

L’esecuzione del comando “semplicemente” registrerà il numero dell’uscita da selezionare alla fine dell’esecuzione, ma questo non determinerà lo stop dello script.

## getExit

Dato che “setExit” non determina lo stop dello script e l’immediata uscita dall’esecuzione, si potrà conoscere l’uscita prescelta attraverso questo comando.

Restituisce un valore integer

$foo = wofoEnv->getExit();

# Indicatori di stato e di fonte della richiesta di esecuzione

la capacità di Flussu di adattare dinamicamente l'interazione in base alla piattaforma di origine e la sua funzionalità di generazione della data arricchiscono l'esperienza utente, rendendo il processo di raccolta delle informazioni più fluido e naturale.

Pertanto in Flussu, gli indicatori di stato di partenza giocano un ruolo cruciale nella definizione dell'interazione con l'utente, permettendo al disegnatore di adattarsi al tipo di applicazione chiamante.

Questa funzionalità è particolarmente utile per determinare il formato e la modalità di presentazione delle richieste all'utente, a seconda che la chiamata provenga da una chat o da una piattaforma “form based”.

Flussu, nella qualità di sistema integrato SOA, è in grado di interagire con Zapier (www.zapier.com) sia in ingresso che in uscita. *Questa funzionalità è descritta in un apposito capitolo di questo manuale*.

## Identificazione della Fonte della Chiamata:

Come anticipato Flussu è progettato per funzionare sia come sistema ad interfaccia chat sia come sistema form (typeform).

Identificare se la chiamata proviene da una applicazione chat o da una interfaccia Typeform o da app esterne come ZAPIER.

## Personalizzazione dell'Interazione in Base alla Piattaforma:

La consapevolezza della fonte della chiamata permette a Flussu di ottimizzare la presentazione delle richieste all'utente.

Le seguenti variabili sono valorizzate con valore booleano al momento dello start del processo:

$isWeb - $isMobile

$isForm

$isTelegram - $isWhatsapp

$isZapier  
$isApp - $isIosApp - $isAndApp (v3.0)

### In Caso di Typeform:

È possibile visualizzare più input contemporaneamente. Ciò significa che se si devono raccogliere dati come nome, cognome ed email, l'utente può inserirli tutti in una volta attraverso tre diversi campi di input.

### In Caso di Chat:

Data la natura sequenziale delle chat, si può richiedere un solo tipo di informazione alla volta. Questo perché le interfacce chat presentano un unico spazio testuale e un unico pulsante di invio.

Pertanto, le richieste devono essere presentate una dopo l'altra.

Nel contesto delle chat, si potrà ulteriormente distinguere tra varie piattaforme come Telegram, WhatsApp, l'app Flussu stessa (v3.0) o una chat HTML/JS.

# Generazione della Data e Formato:

Flussu include la capacità di generare automaticamente etichette per indicare la data corrente, sia nel formato internazionale sia in un formato "esteso" e in lingua.

Il formato esteso presenta non solo il giorno, il mese e l'anno, ma anche il nome del giorno della settimana e l'orario esatto (ore, minuti e secondi).

Questo dettaglio è utile per creare una documentazione precisa e per fornire all'utente informazioni temporali contestualizzate, per esempio quando si accettano i termini della privacy per cui, da necessità GDPR, si devono registrare data e ora dell’accettazione.

wofoEnv->getNow();

wofoEnv->getNow("EN");

wofoEnv->getDateNow();

wofoEnv->getDateNow("IT");

# Generazione di dati casuali

All’interno di un processo può risultare molto utile generare dati casuali, ad esempio per selezionare casualmente delle uscite o per generare codici di verifica o password.

Per questi casi l’environment mette a disposizione 3 comandi, uno per la generazione di numeri interi casuali e due per la generazione “sicura” di codici univoci.

Per “generazione sicura” si intende codici per i quali si sono scelte sequenze che non pongono l’utente nel dubbio.

Per esempio, se chiedessimo all’utente di trascrivere un codice generato come di seguito

**DaF0mO\_l34b**

potrebbe creare incertezza riguardo alla distinzione tra lettere e numeri, come la lettera "O" e lo zero, o la "D" e lo zero, o il numero "1" e la lettera minuscola "l".

Le sequenze predefinite in Flussu sono progettate per evitare tali problemi.

Sebbene i programmatori possano modificare manualmente queste sequenze, il sistema gestisce autonomamente questa funzionalità per coloro che preferiscono non intervenire direttamente.

## getRnd

Il comando restituisce un numero intero casuale a partire dal primo parametro passato e fino al secondo parametro passato.

$foo = wofoEnv->getRnd(0,10);

## generateNewCheckCode

il comando restituisce una stringa contenente un codice casuale composto dal numero di caratteri richiesto.

$foo = wofoEnv->generateNewCheckCode(5);

Il secondo parametro, opzionale, stringa, permette di specificare al comando quale sequenza di caratteri deve essere usata.

$foo = wofoEnv->generateNewCheckCode(4,”ABCabc123$&#”);

## generateNewPassword

Al pari del comando *generateNewCheckCode*, genera un nuovo codice casuale, ma in questo caso la sequenza di caratteri preimpostata, è più ampia

$foo = wofoEnv->generateNewPassword(8);

anche in questo caso, sarà possibile indicare la sequenza che si vuole utilizzare, attraverso il secondo parametro opzionale.

$foo = wofoEnv->generateNewPassword(6,”ABCDE12345”);

# Registratore dei dati

Contrariamente a quanto avviene con le variabili, dove è necessario ricordare il nome per accedere ai dati, il registro di Flussu permette di aggiungere dati senza dover memorizzare il nome della variabile.

Il registro utilizza una struttura di tipo array con coppie etichetta => valore.

Questo approccio facilita la registrazione di informazioni durante l'esecuzione del processo, che possono poi essere estratte in formato JSON. Il formato JSON, una rappresentazione di dati etichettati, è ampiamente utilizzato nei moderni sistemi informativi e di elaborazione.

Questa caratteristica garantisce l'interoperabilità del motore di processo **Flussu** con una vasta gamma di altri sistemi di elaborazione, inclusi database, fogli di calcolo, sistemi gestionali, script, e così via.

## recData

Con questo comando si chiede al registratore di registrare (o aggiornare se esiste) un dato con una determinata etichetta

wofoEnv->recData(*Name*,*Value*);

ovviamente l’etichetta *Name* che il valore *Value* possono essere parametri contenenti dati fissi(A) oppure variabili(B).

A:wofoEnv->recData(“Nome”,”Aldo”);  
B:wofoEnv->recData(“Nome”,$varName);

## getData

Questo comando restituisce il valore registrato con una determinata etichetta

$varValue=wofoEnv->getData($varName);

## getDataJson

Questo comando esporta una stringa nel formato JSON (etichetta/valore) standard, contenente tutti i dati che sono stati accumulati nel registro.

$strJson=wofoEnv->getDataJson();

## setDataJson

Attraverso questo comando è possibile importare, dal formato JSON, più etichette/valori all’interno del registro del server di processo

wofoEnv->setDataJson($strJson);

# Estrattore JSON di array di dati

Per estrarre dati che non sono presenti nel registro del processo usando il formato dati JSON, è possibile usare il comando

## makeJson

con questo comando è possibile trasformare una array in una stringa in formato JSON etichetta/valore.

wofoEnv->makeJson($varArray);

# Esecuzioni esterne

Per arricchire le capacità di interazione con l'ambiente esterno al processo e incrementare l'utilità e l'efficacia del motore **Flussu**, sono stati implementati diversi comandi che consentono esecuzioni "esterne" al motore del processo.

È importante notare che queste esecuzioni esterne vengono attivate solo al termine dello script in un dato *blocco* e il risultato di tali esecuzioni diventa disponibile nel *blocco* immediatamente successivo.

In pratica, il motore di **Flussu** inoltra la richiesta a un modulo esterno non appena lo script all'interno del blocco viene completato. Di conseguenza, l'esecuzione esterna si verifica nel momento in cui lo script è terminato e rilasciato.

Nel blocco successivo, lo script riceverà e incorporerà il risultato dell'esecuzione esterna realizzata precedentemente.

## sendEmail

Com’è facile intuire, attraverso questo comando si potranno inviare email usando tre parametri:

1. l’indirizzo email del destinatario;
2. l’oggetto dell’email;
3. il testo.

L’invio di email che non dovranno finire nello *SPAM*[[5]](#footnote-5) attraversa diversi elementi di verifica e pulizia, uno dei quali prevede che una email contenga il testo del messaggio sia in *HTML* che in testo “normale”[[6]](#footnote-6), pertanto quest’ultimo testo può essere composto in linguaggio **Flussu** *enhanced text*, che assicura il passaggio automatizzato da “plain text“ ad HTML. La definizione dell’*enhanced text* di è presente in un apposito paragrafo successivo.

wofoEnv->sendEmail($toAddress,$subject,$body[,$replyTo]);

Vedi anche, nelle utilità, il comando di verifica formale della correttezza dell’indirizzo email: checkEmailAddress.

## sendPremiumEmail

Attraverso questo comando si potranno inviare email potendo specificare anche il mittente (parametro smtp: mail\_from - <mittente>).

wofoEnv->sendPremiumEmail($to,$subj,$body  
 ,$replyTo,$sendername);

## print2PDF (v2.9)

Flussu può generare un file PDF a partire da un testo (flussu-enhanced) contenuto in una variabile. Il file PDF viene generato su file, pertanto il primo blocco eseguirà il comando e quello successivo riceverà come risultato il filename, incluso il percorso, conenuto in una variabile di cui è indicato il nome in fase di esecuzione del comando.

wofoEnv->printToPdf($title,$txt2Prn,”filePath”);

Un altro comando permette di ottenere lo stesso risultato, ma se il file PDF avesse bisogno di un valore di “testata” e “piede” per tutti i fogli, sarà possibile usare il seguente comando:

wofoEnv->print2PdfwHF($title,$txt2Prn,  
 $flxTxtHead,$flxTxtFoot,  
 ”filePath”);

## sendEmail con fileattach (v2.9)

Seguendo l’esempio precedente dove verrà generato un file PDF il cui filename e path siano inseriti nella variabile $filePath,

$arrAtts=[$attachTitle.".pdf"=>$filePath];  
 wofoEnv->sendEmailwAttaches($to,$subj,$body,$repTo,$arrAtts);

Parimenti si potrà seguire l’invio come email “premium” usando il comando

wofoEnv->sendEmailPremiumwAttaches ( … )

# Integrazione ZAPIER

procedura di integrazione ZAPIER IN e ZAPIER OUT **TBD**

# Integrazione OpenAi

procedura di integrazione OpenAI e ChatAi **TBD**

# Utilità

## checkEmailAddress

Verifica formale del formato di un indirizzo email

Accetta come parametro una stringa e restituisce TRUE o FALSE relativamente alla validità (o meno) formale dell’indirizzo email passato come parametro;

$boolRes=wofoEnv->checkEmailAddress($emailAddr);

## isDisposableEmailAddress

Verifica che l’indirizzo e-mail fornito non faccia parte di server o sistemi che gestiscono email temporanee (es. www.yopmail.com)

Accetta come parametro una stringa e restituisce TRUE o FALSE relativamente alla validità (o meno) formale dell’indirizzo email passato come parametro;

$boolRes=wofoEnv->isDisposableEmailAddress($emailAddr);

## generateQrCode

Generazione di un codice nello standard grafico QR-Code

Accetta come parametro una stringa e visualizza un codice definito nello standard QR-Code che riporta la stringa inviata;

wofoEnv->generateQrCode($qrDataSring);

In pratica genera in memoria un comando che poi inserisce nella lista degli elementi da visualizzare, camuffandolo da “immagine”

## checkCodFiscale

Verifica formale del formato di un codice fiscale italiano

Accetta come parametro una stringa contenente un codice fiscale italiano e, se corretto, oltre il dato di correttezza, restituisce informazioni su sesso e data di nascita;

E’ un comando che viene eseguito esternamente alla sandbox, pertanto i dati saranno disponibili nel blocco successivo.

wofoEnv->checkCodFiscale($cFis,$isGoodVn,$sexVn,$birthdateVn)

Tranne il valore del primo parametro (codice fiscale) che viene passato per valore, gli altri parametri sono passati per “nome”: ciò significa che nel blocco successivo saranno valorizzate delle variabili il cui il “nome” è indicato come stringa nel parametro.

Ciò è necessario per motivi di sicurezza e interoperabilità con la sandBox nella quale sono eseguiti gli script del processo.

Pertanto, all’esecuzione, per esempio di:

wofoEnv->checkCodFiscale($CF,”isGood”,”usrSex”,”usrBday”)

dove $CF contiene il codice fiscale, nel blocco successivo saranno presenti/valorizzate le seguenti variabili:

* + - $isGood
    - $usrSex
    - $usrBDay

Contenti, rispetivamente:

* + - true/false dipendentemente dalla validità o meno del codice fiscale;
    - M/F oppure U se $isGood==false;
    - Data di nascita nella forma YYYY-MM-DD

## Flussu Enhanced Text e Gestione delle risposte

In Flussu, l'arricchimento del testo e la gestione della pagina giocano un ruolo cruciale nella comunicazione efficace e nell'aspetto visivo dei contenuti.

I tag di formattazione "enhanced" consentono una personalizzazione dettagliata del testo negli elementi del blocco, così come la strutturazione della pagina.

Questi tag sono suddivisi in tre categorie: tag per la modifica del testo, tag per la gestione della pagina e tag per elementi multimediali.

## Tag per la Modifica del Testo

Questi tag modificano l'aspetto del testo, come il grassetto, il corsivo, ecc., e sono fondamentali per dare enfasi o evidenziare parti importanti del testo.

{b}: Testo in grassetto. Esempio: {b}**testo**{/b}.

{s}: Testo barrato. Esempio: {s}~~testo~~{/s}.

{u}: Testo sottolineato. Esempio: {u}testo{/u}.

{i}: Testo in corsivo. Esempio: {i}*testo*{/i}.

{w}: Testo di avvertimento (grassetto e rosso). Esempio: {w}**attenzione**{/w}.

{t}: Titolo. Esempio: {t}**Titolo**{/t}.

{h}: Equivalente del tag H1 in HTML. Il testo contenuto all'interno di un tag <h1> è di solito il più grande di tutti i tag di intestazione, rendendolo visivamente dominante sulla pagina. Sono aggiunti automaticamente un ritorno a capo prima e uno dopo.

## Tag per la Gestione della Pagina

Questi tag influenzano la struttura e l'organizzazione del testo, consentendo rientri, separazioni e altri elementi strutturali.

{pl1}, {pl2}, {pl3}: Rientri a sinistra di 5, 10 o 15 caratteri.

{pr1}, {pr2}, {pr3}: Rientri a destra di 5, 10 o 15 caratteri.

{hr}: Linea orizzontale di separazione.

{d}: Riquadro per testo. Esempio:

|  |
| --- |
| TESTO RIQUADRATO |

{pbr}: Salto di pagina.

## Tag per Elementi Multimediali

Per integrare elementi multimediali come link e immagini, Flussu include tag specifici.

{a}: Anchor (link). Esempio: {a}https://sample.com/mypage.html{/a}

{img}: Immagine. Esempio:

{img}style="height:200px;width:auto;max-height:200px" src="https://sample.com/myimage.jpg"{/img}

per inserire un'immagine con attributi di stile definiti.

Questi tag offrono un ampio controllo sul layout, l'aspetto del testo e l'integrazione di elementi multimediali in Flussu, consentendo agli utenti di creare documenti e processi più leggibili, attraenti e funzionali.

Il loro utilizzo appropriato può notevolmente migliorare la comunicazione all'interno dei processi gestiti tramite Flussu.

# Sub-Processi (v2.0)

# Gestione dei sottoprocessi

In questa sezione del manuale, esploriamo la gestione e l'implementazione dei sottoprocessi nel motore Flussu.

Un sottoprocesso in Flussu è essenzialmente un processo standard, disegnato e gestito allo stesso modo di qualsiasi altro processo.

Tuttavia, ci sono alcune specificità da considerare:

## Inizializzazione e Variabili nei Sottoprocessi

I **sottoprocessi** in Flussu iniziano come estensioni di un processo principale.

Di conseguenza, le variabili utilizzate nel processo principale sono già inizializzate quando inizia il sottoprocesso e questo aspetto va attentamente considerato nel *blocco di start* del sottoprocesso.

Esempio: Se una variabile, diciamo $foo, è utilizzata nel sottoprocesso, è fondamentale verificare la sua esistenza prima di inizializzarla.

Questo si può fare utilizzando il seguente codice:

if (!isset($foo)) $foo = "";

Questo controllo assicura che la variabile $foo sia inizializzata correttamente, evitando errori o comportamenti inattesi.

## Integrazione delle Variabili e dello stato

Tutte le variabili create o modificate nel sottoprocesso sono conservate e rese disponibili al processo principale una volta che il sottoprocesso è terminato.

Ciò garantisce la continuità dei dati e la coerenza del flusso di lavoro tra il processo principale e i suoi sottoprocessi

## Gestione dei Blocchi Terminali in Sottoprocessi

Se, durante l'esecuzione di un sottoprocesso, il percorso di esecuzione raggiunge un blocco terminale, l'intero processo (sia il sottoprocesso che il processo principale) si fermerà a quel punto.

Questo comportamento deve essere attentamente considerato durante la progettazione del flusso di lavoro per assicurare che il processo si concluda come previsto

Incorporare questi elementi nella progettazione e gestione dei sottoprocessi in Flussu è cruciale per assicurare un funzionamento fluido e efficiente del sistema.

# Processi multipli (v3.0)

# Gestione dei processi multipli

Dalla v.3.0 è introdotto il concetto di processo multiplo: pari processo in cui cambiano le variabili di start, permettendo di applicare a più clienti lo stesso processo (e le modifiche successive) senza necessariamente generare un processo per ogni cliente.

**TBD**

# Integrazioni Service Oriented (SOA)

# Protocollo di colloquio con sistemi esterni

Una sezione a parte merita il sistema di colloquio con sistemi/programmi esterni.

Come per le esecuzioni esterne, i comandi vengono eseguiti dopo che viene eseguito lo script del blocco. Il risultato è disponibile dal blocco successivo a quello che esegue il comando.

Particolare cura è stata posta nel mantenere il motore di processo indipendente da qualsiasi altro sistema esterno. Questo assicura sia la piena funzionalità del motore in sé, e questo permette di estenderne la funzionalità collegandolo a qualsiasi sistema esterno.

Ciò avviene grazie da un apposito protocollo di colloquio per l’interscambio di informazioni che può essere usato mentre il processo è in esecuzione, potendone pertanto determinare il cambio dello stato.

Ad esempio, se in un processo diventa necessario verificare l’esistenza o il valore di un dato che risiede all’interno di un database di una applicazione terza, ovvero sistemi esterni ovviamente avulsi dal motore Flussu, si potrà usare l’apposito protocollo di colloquio implementando due richieste dal lato dell’applicazione terza, completamente basate su HTTP e JSON.

Ovviamente per preservare la sicurezza, il protocollo è basato su due fasi:

1. Richiesta di autenticazione (autorizzazione via OTP[[7]](#footnote-7));
2. esecuzione di una richiesta usando l’OTP precedentemente generato.

|  |
| --- |
| Fase 1: authentication/authorization  Applicazione  Terza .  getXCmdKey (uid,pass)  Motore  Flussu  **AUTH**  OTP (uuid)  Fase 2: execution  sendXCmdData (OTP)  [result]  Applicazione  terza  Motore  Flussu |
| *Fig.D – Schema di esecuzione del protocollo Flussu* |

## getXCmdKey

Il motore Flussu comunica al sistema esterno di voler eseguire un determinato comando previsto dall’applicazione terza.

Per questo motivo chiede autorizzazione (OTP) inviando dati di autenticazione

wofoEnv->getXCmdKey($srvAdd,$srvCmd,$userId,$password);

I parametri rappresentano:

* $srvAddr: l’indirizzo http/https del server terzo;
* $srvCmd: il nome del comando che vogliamo venga eseguito alla ricezione delle informazioni contenute nel successivo “*sendXCmdData*”;
* $userId: una userId riconosciuta valida dal server terzo;
* $password: la password riconosciuta dal server terzo.

Il server terzo dovrà contenere un connettore di accesso esterno, accettando dati JSON che verranno inviati secondo il seguente schema:

{“command”:[$srvCmd],”userId”:[$userId],“password”:[$password]}

Se i dati di autenticazione sono corretti, il sistema terzo dovrà produrre un codice di autorizzazione univoco (tipicamente un UUID) al quale collegare il comando che dovrà essere eseguito (*vd*: apposito capitolo: Implementazione del protocollo).

Dovrà quindi restituire una risposta in formato JSON:

{"result":"OK","key":[$theOtpKey]}

oppure

{"result":"ERROR","message":[$theMessage]}

L’esecuzione del comando valorizzerà la variabile $XCmdKey che conterrà l’OTP generato dal sistema terzo e verrà essere usata come chiave di autorizzazione nel successivo comando di esecuzione.

## sendXCmdData

Con questo comando si chiede ad un sistema esterno di eseguire un determinato comando, dallo stesso previsto, e restituirne il risultato.

wofoEnv->getXCmdKey($srvAddr,$XCmdKey,$cmdJData,$retVarName);

I parametri rappresentano:

* $srvAddr: l’indirizzo http/https del server terzo;
* $XCmdKey: la chiave OTP generata dal sistema terzo (*vd*: getXCmdKey);
* $cmdJData: i dati da inviare al server terzo per l’esecuzione del suo comando, in formato JSON;
* $retVarName: il “nome” della variabile al cui interno verrà memorizzato il risultato inviato dal server terzo. In questo caso di tratta di un valore stringa (Es.”nomeDellaVariabile”) che si trasformerà poi nella variabile valorizzata (Es. $nomeDellaVariabile). Ciò è necessario per motivi di sicurezza e interoperabilità con la sandBox nella quale sono eseguiti gli script del processo.

L’applicazione terza a sua volta dovrà restituire una risposta in formato JSON:

{"result":"singleResult"}

{"result":"{arrayResult}"}

Il valore del risultato sarà memorizzato nella variabile il cui nome, sotto forma di stringa, è indicato come quarto parametro del comando.

# Implementazione del protocollo di comunicazione

Il protocollo di comunicazione tra Il motore **Flussu** e una applicazione terza, prevede la presenza di un *connettore*, che autorizzi/autentichi le richieste e che poi le esegua e restituisca il risultato atteso dagli script del processo.

Dato un connettore all’indirizzo “https://example.com/wfbcn.php”, il server Flussu eseguirà specifiche richieste come di seguito indicato

## Autenticazione/autorizzazione

La chiamata secondo le specifiche POST[[8]](#footnote-8) del protocollo HTTP, avrà il seguente formato:

https://example.com/wfbcn.php?C=G

il cui contenuto in POST sarà una stringa JSON così composta:

{”command”:[the command name],

”userid”:[the user ID],

”password”:[the password]}

L’applicazione “terza” dovrà

1. Autenticare la richiesta (verifica esistenza e validità della coppia *userId*/*password* inviata)
2. Generare un codice OTP (tipicamente un UUID)
3. Registrare su database:
   1. OTP
   2. Comando richiesto
   3. *UserId* richiedente
   4. [Data della richiesta]
4. Rispondere secondo il seguente schema JSON:
   1. {"result":"OK","key":[$theOTP]}

Oppure

* 1. {"result":"ERROR","message":[$theMessage]}

Nel successiva figura, un esempio minimale di codice PHP per l’implementazione di un connettore di autenticazione che produca un OTP.

|  |
| --- |
| <?php  $Cmd=$\_GET[“C”];  $result=array(“result”=>”bad or unrecognized request”);  if ($Cmd==”G”){  $receivedData=file\_get\_contents('php://input');  $result=$this->getKey($receivedData);  }  die(json\_encode($result));  . . .  function getKey($command){  $uid=$command->userid;  $pwd=$command->password;  $cmd=$command->command;  if ($myObj->authenticate($uid,$pwd)){  $OTP=$myObj->generateOtp($uid,$cmd);  return array("result"=>"OK","key"=>$OTP);  }  return array("result"=>"ERR","message"=>"can’t authenticate”);  } |
| *Fig.E – Implementazione minimale di codice PHP per l’emissione di un OTP* |

## Esecuzione del comando remoto

Considerando la necessità di conoscere lo stato di un valore presente su una applicazione terza, una volta ricevuta l’autorizzazione ad eseguire la richiesta (*vd*: paragrafo precedente), si potrà procedere ad ottener il risultato atteso dall’applicazione remota.

La chiamata, secondo le specifiche POST del protocollo HTTP, avrà il seguente formato:

https://example.com/wfbcn.php?C=E&K=*[theOTP]*

il cui contenuto in POST sarà una stringa JSON così composta:

{ANY Json parameters}

L’applicazione “terza” dovrà

1. Verificare l’esistenza dell’OTP, usando l’OTP come chiave per estrarre il comando che si voleva effettivamente utilizzare (vedi paragrafo precedente);
2. eseguire il comando “estratto”, passando i parametri inviati in questa richiesta;
3. rispondere restituendo un valore atteso, secondo il seguente schema JSON:

{"result": [ANY Json parametrs]}

Il valore del parametro RESULT dei dati in JSON di ritorno, saranno memorizzati nella variabile il cui nome è stato passato come quarto parametro.

Ad esempio:

se avessimo eseguito il seguente comando

wofoEnv->getXCmdKey(“http://xple.com/wfbc.php”,”test”,”{‘test’:’test’}”,”retVar”);

e fosse stato restituito un valore “TRUE” dalla funziona remota, questo sarebbe stato trasferito quale valore contenuto nella variabile $retVar disponibile a partire dal blocco successivo a quello che ha eseguito il comando.

|  |
| --- |
| <?php  $Cmd=$\_GET[“C”];  $result=array(“result”=>”bad or unrecognized command”);  if ($Cmd==”E”){  $OTPkey=$\_GET[“K”];  $Exec=$this->getCommand($OTPkey);  $receivedData=file\_get\_contents('php://input');  if ($Exec==”test”)  $result=$this->test($receivedData);  else  $result->result=”bad or unrecognized OTP”);  }  die(json\_encode($result));  . . .  function **test**($parameters){  $res= ($parameters->test==”test”);  return array(“result”=>$res);  } |
| *Fig.F – Implementazione minimale di codice PHP per l’esecuzione di un comando remoto* |

# Connettore API / Esecuzione chatbot

Il protocollo di esecuzione del processo è, come per il protocollo di editing e rappresentazione, basato su http sull’interscambio di dati in formato JSON.

# Protocollo di esecuzione

Il motore di processo si occupa di tutto il lavoro e selezione dei blocchi e di elaborazione dei dati all’interno.

Come anticipato nel capitolo relativo alla programmazione, il motore di workflow genera una “sessione” per ogni esecuzione di processo.

La sessione contiene tutte le informazioni e le variabili necessarie al corretto e programmato funzionamento del workflow.

Pertanto, per utilizzare il motore di database sarà sufficiente eseguire una chiamata al connettore di esecuzione passando il codice del workflow e la lingua in cui si desidera interagire.

/flussuscript/flussueng.php

Questo connettore API ha tre modalità funzionali,

1. restituire le “informazioni” sul processo, quali titolo, lingue considerate e lingua di default, spesso necessarie alla prima esecuzione o alla preparazione di una interfaccia;
2. inserire dei dati di stato da registrare prima dell’esecuzione, funzionalità necessaria ad eseguire un workflow potendo predeterminarne lo stato, necessario quando un workflow salta l’esecuzione verso un altro workflow oppure quando è necessario predeterminare alcune richieste. Per esempio quando si deve inserire un valore da visualizzare “gentile cliente (nome passato) deve pagare (valore passato) euro, come vuole procedere?” ed iniziare così il flusso d’esecuzione. Questa modalità è discussa nel capitolo successivo “Esecuzione arbitraria”;
3. Attraverso questo connettore si eseguiranno tutti i singoli passi del processo.

Questo connettore viene utilizzato passando alcune informazioni sotto forma di variabili in http/post o in http/get. Il parametro sempre necessario è il codice del workflow (WID), gli altri parametri dipendono dalla modalità operativa.

## Modalità informativa

Per ottenere informazioni su un workflow, come ad esempio il titolo, le lingue supportate e la lingua di default, si dovrà richieredere il “command” di start.

/flussuscript/flussueng.php?WID=[wb2d3778a7]&CMD=start

Vengono così restituiti, in formato JSON, i seguenti dati:

|  |
| --- |
| tit :"Titolo del processo"  langs :"IT,EN,FR,DE" *(lingue supportate)*  def\_lang :"IT" *(lingua di default)* |

## Modalità esecutiva

Trasmettendo una informazione rappresentata in JSON che contenga i seguenti elementi quando necessari:

* + - * WID: workflow-Id,
      * LNG: lingua,
      * SID: session-Id,
      * BID: block-Id,
      * TRM: termini/valori

## Startup dallo start block

Lo startup di un workflow si ottiene passando i valori di workflow id e, opzionalmente, quelli relativi alla lingua. Se quest’ultimo dato non è usato, verrà usata la lingua di default.

/flussuscript/flussueng.php?WID=[wb2ddd575b13778a7]

In questo modo il motore eseguirà le prime operazioni e restituirà un dato JSON che contiene l’ID della sessione e gli elementi da visualizzare.

In questo caso sarà eseguito il workflow usando la lingua di default e considerando lo startup da interfaccia web.

Aggiungendo il parametro LNG si potrà eseguire il processo in una lingua supportata diversa dalla lingua di default, inoltre usando il parametro APP si potrà registrare nel sistema statistico che la chiamata arriva da una determinata APP (es. TGM (Telegram), MSG (FB/messenger), (WZP) Whastapp, ecc.

/flussuscript/flussueng.php?WID=[wb2d … 8a7]&LNG=IT&APP=WZP

In risposta al comando di startup, il connettore API restituirà una risposta in formato JSON come quella di seguito

|  |
| --- |
| "sid":"9ca76007-452f-f3ce-2993-5061085365d1",  "bid":"06646006-2c90-05f1-c714-41582494d224",  "elms":{  "L$1":["Come ti chiami?",""],  "ITT$name":["",""],  "ITB$0":["Ok",""]  } |
| *Schema 6 – risposta JSON per lo startup del processo di test* |

Nello specifico :

1. il primo elemento “L$1” è una Label il cui valore è “Come ti chiami?” e il cui class è “”.
2. Il secondo è una TextBox il cui “nome per il valore di ritorno” è $name. La label legata a questo elemento di input è “” e il class è “”.
3. Il terzo è un Button ITB il cui collegamento in caso di click è l’uscita $0 (uscita 0). Anche in questo caso il class da applicare è “”.

In questo caso il programma client si deve occupare di rappresentare, nel modo che preferisce, il dato che il motore gli richiede, quindi recuperare (eventualmente) gli elementi che servono per interagire con l’utente (valori di textbox, select list o pulsanti premuti) ed inviarle al server.

### Ad esempio

|  |
| --- |
|  |
| *Fig.J – Interfaccia generata con i dati dell’esempio* |

E’ importante prendere nota che sia Block-ID sia Session-ID sono valori che, come anticipato, sarà necessario allegare al messaggio di risposta al motore di workflow.

Pertanto al connettore /flussuscript/flussueng.php veranno passati ***in POST*** i seguenti valori

|  |
| --- |
| "WID" : "[wb2ddd575b13778a7]"  "SID" : "b4f66007-6fba-ac56-0263-48216956cef1"  "BID" : "06646006-2c90-05f1-c714-41582494d224"  "LNG" : "IT"  "TRM" : "{"$ex!0":"Ok","$name":"Aldo"}" |
| *Schema 7 – invio dei dati del processo di test* |

## Termini

Nello specifico il valore relativo ai **terms** (TRM) è così composto:

* + $ex!0= è stata selezionato il bottone con l’uscita 0;
    - il valore del testo sul pulsante era “OK”
  + $name è l’etichetta della textbox, il cui valore passato è “Aldo”.

Parimenti per le liste select, verrà passato un nome al quale dovrà essere accoppiato il valore del “value”.

"ITS$selScelta":[{"123":"esempio","124":"prova","test":"test text"}]

In questo caso se l’utente avesse scelto il secondo elemento della lista (visualizzato “prova” valore “124”) il TRM in risposta sarebbe stato il seguente:

"TRM" : "{"$selScelta":"124"}"

## Restituzione parametri

Può capitare di chiedersi perché sia necessario all’interno della risposta passare gli ID di workflow, sessione e blocco.

Tecnicamente sarebbe necessaria solo la sessione, ma potendo passare anche l’id del blocco, sarà possibile, lato client, implementare un tasto “torna indietro” e così inviare l’ID del blocco precedente. Questo farà si che il motore esegua nuovamente il blocco dando l’impressione di tornare indietro.

L’ID del workflow in realtà è una chiave di sicurezza che sarà spiegata nel capitolo relativo a sicurezza, utenti, chiavi di attivazione, ecc.

# Startup da un blocco arbitrario

Vista la sintassi del connettore API, indicata nel capitolo precedente, è facilmente intuibile che sia possibile eseguire un workflow a partire da un blocco arbitrariamente scelto tra quelli del processo, passando tutti i parametri e valori dei termini necessari. In questo caso si dovrà specificare un valore “arb\_start” nel “session id”.

I valori mandatori sono il workflow id, la lingua ed il block\_id.

/flussuscript/flussueng.php?

WID=[workflow-id]

&LNG=[language-code]

&BID=[block-id]

&SID=arb\_start

[&TRM=…]

In questo modo il motore inizierà l’esecuzione del workflow a partire dal blocco indicato, registrando i valori dei termini indicati.

E’ consigliabile predisporre un apposito blocco di “ingresso” nel workflow che definisca anche eventualmente parametri di default, così come verrà spiegato in dettaglio nell’apposito capitolo successivo.

## Modalità assegnazione valori arbitrari ai parametri

Molto utile in caso di esecuzione arbitraria, la modalità di assegnazione arbitraria di valori ai parametri, è spesso necessaria.

Consideriamo di avere chiesto di iniziare il processo partendo da un blocco che si aspetta un valore di stato da un input di un utente o dal risultato dell’esecuzione di una esecuzione esterna.

Per esempio il terzo blocco del seguente processo, che dopo avere acquisito il valore della variabile ***$nome***, la usi per salutare l’utente.

|  |
| --- |
|  |

Come anticipato, attraverso l’esecuzione arbitraria si potrà decidere di eseguire direttamente il terzo blocco si questo processo d’esempio, ma dato che la variabile ***$nome*** sarà vuota, attraverso l’assegnazione di valori arbitrari ai parametri, possiamo fare in modo che l’esecuzione arbitraria del terzo blocco venga eseguita valorizzandone dall’esterno, in chiamata, anche le variabili.

## Passaggio di valori di startup/arbitrari

Nel momento in cui viene richiamato un blocco, ovvero allo start-up di un processo, è possibile passare parametri arbitrari che saranno poi memorizzati tra le variabili disponibili nel processo.

In questo modo sarà ad esempio possibile utilizzare lo stesso script workflow per differenti siti, dove allo start-up viene indicato il nome del sito, l’indirizzo email, un testo di saluto, ecc. che viene cambiato utilizzando questa tecnica, al momento dell’esecuzione.

**TBD.**

# Esecuzione chatbot da CHAT APP standard

# Esecuzione da chat app standard

Flussu, in quanto sistema SOA può essere usato attraverso il sistema di API di APP di chat standard come Telegram o Whatsapp.

## Introduzione ai Bot di Telegram e l'Integrazione con Flussu

Per comprendere come Flussu può eseguire i suoi processi all'interno di Telegram, è fondamentale avere una conoscenza di base del funzionamento dei bot su questa piattaforma di messaggistica.

### Cosa Sono i Bot di Telegram

I bot di Telegram sono applicazioni di terze parti che funzionano all'interno di Telegram, operano su server esterni e non richiedono un'app aggiuntiva; interagiscono con gli utenti attraverso messaggi all'interno dell'app Telegram.

Vengono creati e configurati utilizzando "***BotFather***", un bot speciale fornito da Telegram che aiuta gli sviluppatori a creare e gestire i propri bot.

Con **BotFather**, è possibile assegnare nomi ai bot, impostare comandi e ottenere un token API unico per il bot.

### Integrazione di Flussu con Telegram

Flussu sfrutta questa tecnologia per eseguire processi all'interno dell'ecosistema Telegram.

Utilizzando un'interfaccia chiamata "tg.flu", Flussu stabilisce una connessione tra il server cloud di Telegram e il proprio server, indicato come "srvXX.flu.lu".

Questa integrazione permette a Flussu di "parlare" con Telegram, trasformando i bot di Telegram in interfacce per eseguire processi Flussu.

Gli utenti possono interagire con questi processi direttamente attraverso la chat di Telegram.

### Come Funziona nella Pratica

Quando un utente invia un comando a un bot Telegram collegato a Flussu, il messaggio viene trasferito tramite ***tg.flu*** al server di Flussu.

|  |
| --- |
| **FLUSSU SERVER**  **TG**  **FLU**  Telegram  APP  **TELEGRAM SERVER**  Server Flussu (srv01.flu.lu)  Service di Interfaccia (tg.flu.lu) |
| *Sistema di funzionamento di un processo Flussu dentro Telegram* |

Flussu processa la richiesta e invia una risposta indietro al bot di Telegram, che a sua volta la visualizza all'utente nella chat.

Questo permette un'interazione fluida e naturale, in cui gli utenti di Telegram possono avviare, gestire e interagire con i processi di Flussu direttamente all'interno dell'app.

### Vantaggi dell'Integrazione di Flussu con Telegram

* **Accessibilità**: Gli utenti possono accedere ai processi Flussu direttamente da Telegram, una piattaforma che molti già utilizzano quotidianamente.
* **Interfaccia Familiare**: Utilizzare i bot di Telegram per i processi Flussu rende l'esperienza utente più semplice e familiare.
* **Comunicazione in Tempo Reale**: La possibilità di ricevere risposte e aggiornamenti in tempo reale all'interno di Telegram migliora l'efficienza e l'efficacia della comunicazione.

In sintesi, l'integrazione di Flussu con Telegram apre nuove strade per la gestione e l'esecuzione di processi, sfruttando la popolarità e la facilità d'uso di Telegram per offrire un'esperienza utente migliorata e più accessibile.

## Bot Father

Ecco una guida passo-passo per acquisire i dati necessari per il tuo bot:

**Passo 1: Avvia BotFather**

* Apri Telegram e cerca "BotFather", che è il bot ufficiale di Telegram per creare e gestire altri bot.
* Avvia una chat con BotFather cliccando su "Start".

**Passo 2: Crea un Nuovo Bot**

* Digita il comando /newbot.
* BotFather ti chiederà di inserire un nome per il tuo bot. Questo è il nome che verrà visualizzato agli utenti.
* Successivamente, ti verrà chiesto di inserire un username per il bot, che deve terminare in "bot" (ad esempio, "mioesempio\_bot").

**Passo 3: Ricevi il Token API**

* Dopo aver configurato nome e username, BotFather genererà un token API unico per il tuo bot. Questo token è essenziale per interagire con l'API di Telegram e controllare il bot.
* Salva questo token in un luogo sicuro poiché sarà necessario per configurare l'interfaccia tra il tuo bot e Flussu.
* Questi dati vanno inseriti nell’apposito spazio che è disponibile nell’interfaccia di configurazione del processo, come indicato:

|  |
| --- |
|  |
| *Inserimento dei dati del bot Telegram nell’interfaccia di configurazione del processo* |

**Passo 4: Configura il Bot**

Ora che hai il token, puoi configurare il tuo bot secondo le tue necessità. BotFather offre opzioni per impostare descrizioni personalizzate, comandi e altro.

Per accedere alle opzioni di configurazione, utilizza in Both Father il comando /mybots e seleziona il tuo bot dall'elenco.

**Passo 5: Implementa il Bot nel Tuo Sistema**

Con il token API in mano, puoi ora implementare il tuo bot in Flussu o in qualsiasi altra piattaforma che desideri integrare.

Utilizza il token API per collegare il bot alle funzionalità di Flussu, permettendo così al bot di gestire processi, inviare notifiche o rispondere a comandi tramite Telegram.

Puoi sempre tornare a BotFather per modificare le impostazioni del tuo bot in futuro.

## Introduzione alle chat WhatsApp e l'Integrazione con Flussu

***Questo capitolo considera una funzionalità ancora in fase di definizione. Il testo qui inserito può essere modificato anche totalmente prima del rilascio finale (v3.0)***

L'integrazione di Flussu con WhatsApp presenta alcune differenze chiave rispetto a quella con Telegram, principalmente a causa della necessità di associare un numero di telefono e del processo di configurazione gestito attraverso META (precedentemente conosciuta come Facebook Inc.), che possiede WhatsApp.

Di seguito, una panoramica generale di come Flussu può essere integrato con WhatsApp e delle sfide correlate.

### Necessità di un Numero di Telefono

A differenza di Telegram, dove i bot vengono creati e gestiti tramite BotFather senza la necessità di un numero di telefono, l'integrazione con WhatsApp richiede l'assegnazione di un numero telefonico unico al bot.

Questo numero fungerà da punto di contatto tra gli utenti WhatsApp e il bot Flussu.

### Configurazione Tramite META

La configurazione del bot di WhatsApp per l'integrazione con Flussu passa attraverso la piattaforma di META.

Questo processo può essere più complesso rispetto a Telegram, richiedendo la registrazione e la configurazione attraverso i sistemi di META, che gestisce le API di WhatsApp.

|  |
| --- |
| **FLUSSU SERVER**  **WA**  **FLU**  Telegram  APP  **TELEGRAM SERVER**  Service di Interfaccia (wa.flu.lu)  Server Flussu (srv01.flu.lu) |
| *Sistema di funzionamento di un processo Flussu dentro WhatsApp* |

### Passaggi della Configurazione

**Registrazione del Numero**

Inizialmente, è necessario registrare un numero di telefono che sarà associato al bot di WhatsApp. Questo numero deve essere esclusivo e non usato per altri account WhatsApp.

**Configurazione delle API**

Dopo la registrazione, si procede con la configurazione delle API di WhatsApp attraverso la piattaforma di META. Questo include l'ottenimento di un token di accesso e la configurazione delle impostazioni di sicurezza e privacy.

**Collegamento con Flussu**

Una volta completata la configurazione su META, il bot di WhatsApp può essere collegato a Flussu, permettendo il trasferimento delle informazioni tra WhatsApp e il server di Flussu.

### Sfide e Considerazioni

La necessità di un numero di telefono e il passaggio attraverso la piattaforma di META rendono la configurazione del bot di WhatsApp più complessa rispetto a quella di Telegram.

Considerando le politiche di META e di WhatsApp, è fondamentale assicurarsi che tutte le configurazioni rispettino le normative sulla privacy e sulla sicurezza dei dati.

### Vantaggi dell'Integrazione con WhatsApp

Nonostante la maggiore complessità, l'integrazione di Flussu con WhatsApp offre l'accesso a una vasta base di utenti.

WhatsApp è una delle piattaforme di messaggistica più diffuse al mondo, rendendo questa integrazione particolarmente utile per raggiungere un pubblico ampio e variegato.

In conclusione, mentre l'integrazione di Flussu con WhatsApp presenta maggiori sfide, soprattutto in termini di configurazione e gestione della privacy, i benefici derivanti dall'accesso a una piattaforma di messaggistica globale come WhatsApp sono significativi, ampliando le possibilità di interazione e di engagement con gli utenti.

# Dati statistici

# Dati statistici

Lorem ipsum **TBD**

## Integrazione

Lorem ipsum **TBD**

## Limiti

Lorem ipsum **TBD**

# Sessioni d’uso

# Sessioni d’uso

Lorem ipsum **TBD**

## Integrazione

Lorem ipsum **TBD**

## Limiti

Lorem ipsum **TBD**

# Flussu APP Android+iOs (v3.0)

# Flussu App

Dalla v3.0 Flussu avrà una propria APP che funzionerà sia su sistema Android che su iOs

## Integrazione

Lorem ipsum **TBD**

## Limiti

Lorem ipsum **TBD**

# **Programmazione avanzata**:

# motore server di Flussu

**Quella di seguito descritta è la v1.0 🡪 documentare la v2.0**

***DISCLAIMER***

*Questa parte di manuale considera la programmazione del codice alla base del motore workflow di Flussu, ovvero la possibilità di eseguire una “personalizzazione approfondita”.*

*Ovviamente ciò renderà impossibile un aggiornamento automatico del software server, poiché la versione modificata sarà considerata come un FORK della versione di partenza.*

*Per questa ragione si consiglia di realizzare personalizzazioni su file esterni e codice richiamato come classi/librerie esterne, così da rendere meno gravose le operazioni di re-integrazione con versioni successive di Flussu.*

*Ovviamente il programmatore si assume la responsabilità del buon funzionamento del motore in quanto Mille Isole SRL non potrà rispondere di bug e malfunzionamenti di una versione di software alla stessa sconosciuta.*

Il server di processo **Flussu** può essere esteso a piacimento facendone di fatto il più duttile server di processo esistente sul mercato: può essere applicato e/o integrato in qualsiasi soluzione terza.

# Classe Environment

Il codice eseguito in sandbox, istanzia, come prima operazione, la classe environment attraverso la quale vengono eseguiti comandi inseriti dal programmatore nei blocchi del processo.

Questa classe è uno script interpretato PHP che risiede in un apposito file.

Questo assicura all’utilizzatore di poter estendere a piacimento le funzionalità della sandbox di **Flussu**: qualsiasi funzionalità aggiunta alla classe “environment” può essere direttamente eseguita dal codice nella sandbox, ovvero in ogni blocco del processo.

## Elementi del server

Il server è realizzato secondo i dettami dell’OOP[[9]](#footnote-9) e, nella versione 1.0 è stato utilizzato il linguaggio di programmazione PHP. Dato che PHP rientra nei dettami dell’OOP solo forzatamente, per il prossimo futuro sono previsti porting in Java (Spring) e in C#)

La “visualizzazione”, alla v1.0, è resa in HTML(5) / CSS(3) e javascript (Ecmascript>9), utilizzando il framework Boostrap4 per assicurare la corretta visualizzazione su tutti i media, dal PC allo smartphone, senza che sia necessario scrivere codice apposta.

Tutti gli elementi di codice che compongono la versione 1.0 sono scritti in linguaggio interpretato, sicchè sono liberamente modificabili dall’utente (a suo rischio e pericolo).

## Classi

Come anticipato, il server di processo è diviso in classi/oggetti

|  |
| --- |
|  |
| *Fig.G – Elementi del server Flussu in linguaggio PHP* |

## Flussut\Flussut

Oggetto di collegamento tra il server e:

* lo strato database
* il server http

Necessita quindi della gestione del collegamento al database e delle funzionalità necessarie a recuperare i dati dalle richieste che arrivano via http.

Nel caso dell’implementazione 1.0, è ospitato in un ambiente “MilleIsole/Eskuu” che interfaccia l’ambiente server Linux (CentOs) con server http Apache e database MariaDB (nelle versioni successive verranno realizzati strati proprietari per renderlo indipendente dalla piattaforma).

## Flussut\WofoCmd

E’ l’oggetto che si occupa di eseguire i comandi esterni.

Nel caso in cui siano state aggiunte funzionalità al server di processo, in questa classe dovrebbero essere scritte le implementazioni funzionali.

Nel Flussu v1.0 in questa classe sono presenti le seguenti funzionalità:

* phpMailerSendMail

Invio email utilizzando la libreria open source PhpMailer

* localSendMail

Invio email colloquiando direttamente con il server SMTP installato sullo stesso server dove viene eseguito il motore di processo. E’ il metodo preferito perché assicura la corretta e spedita esecuzione

* getKey/ getCmd

Sono le funzionalità alla base del “protocollo di comunicazione” (*vd*: apposito capitolo precedente)

* execRemoteCommand

Funzionalità di servizio per i comandi getKey e getCmd. Si occupa dell’effettivo colloquio http.

* htmlSanitize/ textSanitize

Sono le funzioni alla base del **Flussu** *enhanced text.*

## Flussut\WofoEnv

E’ l’oggetto a servizio della sandbox di Flussut.

In questa classe sono presenti tutte le funzionalità descritte nel capitolo relativo alla programmazione dei blocchi.

Nel caso in cui siano state aggiunte funzionalità al server di processo, in questa classe dovrebbero essere scritte le implementazioni funzionali.

Nel Flussu v1.0 in questa classe sono presenti le seguenti funzionalità:

* phpMailerSendMail

Invio email utilizzando la libreria open source PhpMailer

## Flussut\WofoSess

E’ l’oggetto che si occupa di gestire una sessione di processo.

E’ responsabile dello stato del processo, delle informazioni sulla lingua prescelte e di tutte le operazioni che avvengono in memoria e poi persistite su database.

Contiene anche un registratore dei LOG.

## Flussut\WofoWork

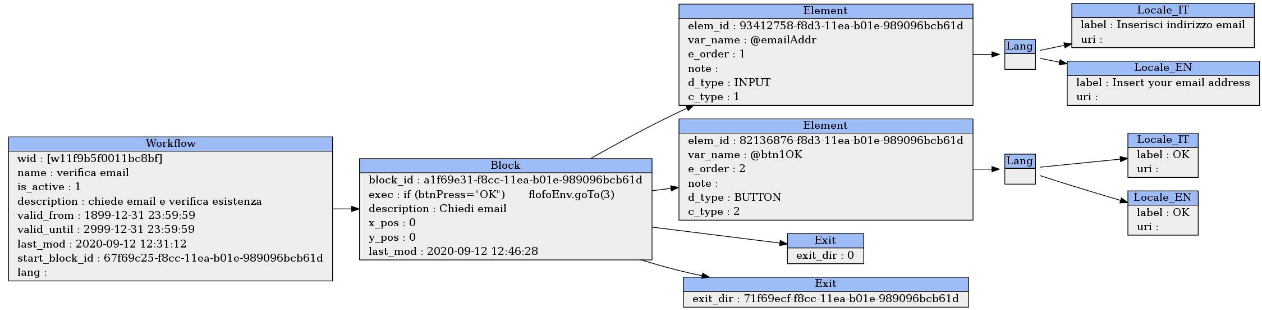
E’ il motore del workflow, l’oggetto che si occupa di scegliere i blocchi, estrarre le informazioni, eseguire gli script.

Work, insieme a Sess e al database, compongono il cuore del sistema di processo.

# Editing e rappresentazione dei processi

La rappresentazione dei dati dei processi è basata sullo standard JSON e il sistema di editing è totalmente basato su http (get e post) attraverso un apposito connettore.

# Forma dei dati

Il formato di rappresentazione dei dati del workflow è quello dell’immagine in fig.H

|  |
| --- |
|  |
| *Fig.H – Disegno del database operativo di Flussu (versione 3)* |

# Metodo d’uso

Il connettore “flow.php” si aspetta i seguenti parametri:

C 🡪 command

[element] 🡪 l’elemento oggetto del comando.

# Comandi disponibili

## GET Workflow List

Con la seguente richiesta get viene restituita la lista di workflow presenti sul server

flow.php?C=L

Sui otterrà una array di workflow:

|  |  |  |
| --- | --- | --- |
| Parametro | Valore (di esempio) | descrizione |
| wid | "[w11f9b5f0011bc8bf]" | Workflow ID |
| Name | "Verifica Email" | Nome/titolo |
| description | "Controlla esistenza email" |  |
| userId | "0" | Utente |
| is\_active | "1" | 1=attivo  0=non attivo |
| supp\_langs | "IT,EN,FR,DE" | Lingue supportate |
| valid\_from | "1899-12-31 23:59:59" | Data di inizio e |
| valid\_until | "2999-12-31 23:59:59" | fine validità |
| last\_mod | "2021-01-08 19:08:17" | Data Ultima modifica |
| *Schema 1 – rappresentazione JSON del workflow* | | |

## Create NEW workflow

Con la seguente richiesta post viene creato un nuovo workflow

flow.php?C=C&N=(nome workflow)

[POST] 🡨 (json)

I parametri da inviare in Json sono:

|  |  |  |
| --- | --- | --- |
| Parametro | Valore (di esempio) | descrizione |
| description | "workflow di test" | Descrizione |
| supp\_langs | "IT,EN,FR" | Lingue supportate |
| Lang | "EN" | Lingua di default |
| *Schema 2 – rappresentazione JSON del workflow* | | |

## GET Workflow

Con la seguente richiesta get viene restituito l’intero workflow con gli elementi nella lingua di default

flow.php?C=G&WID=(codice workflow)

Con la seguente richiesta get viene restituito l’intero workflow nella lingua indicata

flow.php?C=G&WID=(codice workflow)&LNG=(codice lingua)

### Editing

in caso di editing vengono restituiti tutte le etichette in lingua per tutti gli elementi tutte le lingue configurate.

flow.php?C=E&WID=(codice workflow)

Il risultato del get workflow conterrà un elemento workflov (v. schema1) a cui si aggiunge il dato “start-block-id” che identifica il block-id di start, e un’array di blocchi (v. schema2)

## Update workflow

Con la seguente richiesta in cui il dato JSON deve essere passato in POST, il server prenderà in carico l’interno disegno del workflow e si occuperà di fare l’update di tutto il database

flow.php?C=U&WID=(codice workflow)

[POST] 🡨 (json)

## GET block

Con la seguente richiesta get viene restituito l’intero blocco identificato dal suo UUID, nella lingua di default

flow.php?C=G&UUID=(codice blocco)

Con la seguente richiesta get viene restituito l’intero blocco nella lingua indicata

flow.php?C=G&UUID=(codice blocco)&LNG=(codice lingua)

Ogni blocco conterrà uno o più elementi e almeno due uscite. Nello Schema 4 sono visibili gli elementi con tutte le lingue e nello Schema 5 le uscite. A seguire nello schema 6 è visibile lo schema di un intero workflow.

### Editing

in caso di editing vengono restituiti tutte le etichette in lingua per tutti gli elementi tutte le lingue configurate.

flow.php?C=E&UUID=(codice blocco)

|  |  |  |
| --- | --- | --- |
| Parametro | Valore (di esempio) | descrizione |
| block\_id | "ddb84c75-e46d-4da3-98f6-954beb499d02" | Block ID |
| type | "" | Tipo blocco “S=subworkflow” |
| exec | "wofoEnv->Init();" | Codice da eseguire |
| is\_start | "0" | 1=start block |
| description | "Nome blocco" | Descrizione |
| x\_pos | "-1596" | Editor: posizione X |
| y\_pos | "1" | Editor: posizione Y |
| last\_mod | "2021-01-15 20:06:12" | Data Ultima modifica |
| *Schema 3– rappresentazione JSON dei dati del blocco* | | |

Update Block

Con la seguente richiesta in cui il dato JSON deve essere passato in POST, il server prenderà in carico l’interno disegno del workflow e si occuperà di fare l’update di tutto il database

flow.php?C=U&UUID=(codice blocco)

[POST] 🡨 (json)

restituisce un “result” json

{“result”:”OK”}

{“result”:”ERR:##-Descrizione”}

|  |  |  |  |
| --- | --- | --- | --- |
| Parametro | | Valore (di esempio) | descrizione |
| elem\_id | | "0f6c5ffa-8e18-84b6-b892-805425d5a9ad" | Element ID |
| var\_name | | "$userName" | Etichetta della variabile |
| e\_order | | "1" | Ordine di apparizione |
| c\_type | | "1" | Codice tipo |
| note | | "" | Note |
| d\_type | | "INPUT" | Descrizione tipo |
| css | | "" | Nome della “class” da applicare all’elemento |
| exit\_num | | "" | Uscita collegata (se button) |
| langs | |  | Array di elementi |
| DE | label | "Wie ist dein Vorname?" | Label in lingua |
|  | uri | "" | Uri per la lingua |
| EN | label | "What's your first name?" | Label in lingua |
|  | uri | "" | Uri per la lingua |
| FR | label | "Quel est votre prénom?" | Label in lingua |
|  | uri | "" | Uri per la lingua |
| IT | label | "Qual'è il tuo nome?" | Label in lingua |
|  | uri | "" | Uri per la lingua |
| *Schema 4 – rappresentazione JSON degli elementi di un blocco in modalità “edit” (tutte le lingue)* | | | |

|  |  |  |
| --- | --- | --- |
| Parametro | Valore (di esempio) | descrizione |
| exit\_dir | "ddb84c75-e46d-4da3-98f6-954beb499d02" | Zero o Block-ID a cui punta l’uscita |
| *Schema 5 – rappresentazione JSON dei dati di una uscita* | | |

## Delete Block

Con la seguente richiesta viene eliminato un blocco e fatto l’update dell’intero database

flow.php?C=D&UUID=(codice blocco)

restituisce un “result” json

{“result”:”OK”}

{“result”:”ERR:##-Descrizione”}

## Duplicate Block

Con la seguente richiesta viene duplicato un blocco includendo i suoi elementi, il loro valori e le connessioni d’uscita

flow.php?C=P&UUID=(codice blocco)

restituisce un “result” json

{“result”:[new block id]}

{“result”:”ERR:##-Descrizione”}

# Trasferimento Workflow

Sebbene Flussu venga usato come servizio remoto da diversi siti, l’architettura di prevede che il server Flussu possa essere condiviso o possa essere bilanciato o usato in ambienti di microservizi, ecc.

Attualmente esiste un server di Mille Isole che produce ed edita i workflow, ma nel caso in cui si sia predisposto un ambiente di editing diverso dall’ambiente di produzione, ma anche ambienti remoti per clienti che preferiscono aver gestiti i propri workflow su server privati, sarà possibile trasferire un workflow tra server.

E’ esclusivamente necessario che vi sia un “ambiente di esecuzione” Flussu sul server remoto al quale trasferire il workflow.

E’ ovviamente possibile escludere le routine ed i programmi di editing dall’ambiente remoto o di sola esecuzione.

Per trasferire un workflow al server remoto sarà necessario eseguire il seguente comando:

flow.php?C=DEPLOY&WID=[wflow-id]&TO=www.server.com/flow.php

Dove “TO” punta ad un server/connettore di ricezione del workflow (nel caso di questo manuale ad un altro script flow.php).

Il server di editing invierà una richiesta “RD” (Receive Deploy) al server remoto, passando in post il workflow in formato json.

Nel server ricevente verrà implementato tutto il nuovo workflow nel database remoto, creandolo se non esiste o rigenerandone uno esistente (fare molta attenzione alla sovrascrittura!!!).

*TBD*

# Estrazione schema di un workflow d’esempio

Il seguente workflow d’esempio è composto da tre blocchi, di cui uno di start.

Il processo considera due lingue, italiano e inglese.

|  |
| --- |
|  |
| *Fig.I – Disegno di un processo d’esempio – PROCESSI/ENTITA’* |

Il blocco di start ha una espressione di exec, nessun element e una uscita che punta al secondo blocco.

Il secondo blocco ha 3 element, una label, una textbox e un button che punta all’uscita 0, la quale punta all’ultimo dei tre blocchi.

L’ultimo blocco ha una label e non ha uscite (blocco di end).

Un processo minimo/d’esempio, come disegnato in fig.H, produce il seguente JSON:

|  |
| --- |
| "workflow":[  {  "wid":"[wb2ddd575b13778a7]",  "name":"example",  "description":"workflow d'esempio",  "userId":"0",  "is\_active":"1",  "supp\_langs":"IT,EN",  "lang":"EN",  "valid\_from":"1899-12-31 23:59:59",  "valid\_until":"2100-01-01 00:00:00",  "last\_mod":"2021-01-19 15:17:34",  "start\_block\_id":"5f9c6006-9feb-bb4b-6396-73401486d407",  "blocks":[  {  "block\_id":"5f9c6006-9feb-bb4b-6396-73401486d407",  "type":"",  "exec":"wofoEnv->init();\r\n",  "is\_start":"1",  "description":"START BLOCK",  "x\_pos":"113",  "y\_pos":"93",  "last\_mod":"2021-01-19 15:57:04",  "elements":[],  "exits":[  {"exit\_dir":"06646006-2c90-05f1-c714-41582494d224"},  {"exit\_dir":"0"}  ]  },{  "block\_id":"06646006-2c90-05f1-c714-41582494d224",  "type":"",  "exec":"",  "is\_start":"0",  "description":"Chiedi nome",  "x\_pos":"478",  "y\_pos":"224",  "last\_mod":"2021-01-19 15:58:15",  "elements":[  {  "elem\_id":"d5dd6006-2fe6-6fa3-3298-4280754913de",  "var\_name":"",  "e\_order":"1",  "note":"",  "d\_type":"LABEL",  "css":"",  "c\_type":"0",  "exit\_num":"",  "langs":{  "EN":{"label":"What's your name?","uri":""},  "IT":{"label":"Come ti chiami?","uri":""}  }  },{  "elem\_id":"6da16006-3051-1e86-0889-109007541122",  "var\_name":"$name",  "e\_order":"2",  "note":"",  "d\_type":"INPUT",  "css":"",  "c\_type":"1",  "exit\_num":"",  "langs":{  "EN":{"label":"","uri":""},  "IT":{"label":"","uri":""}  }  },{  "elem\_id":"ade34be1-99c8-43f7-a5ab-cbe1d228dee1",  "var\_name":"$btnOk",  "e\_order":"3",  "note":"",  "d\_type":"BUTTON",  "css":"",  "c\_type":"2",  "exit\_num":"0",  "langs":{  "EN":{"label":"Ok","uri":""},  "IT":{"label":"Ok","uri":""}  }  }  ],  "exits":[  {"exit\_dir":"ab736006-3172-261d-e301-985391782bd6"},  {"exit\_dir":"0"}  ]  },{  "block\_id":"ab736006-3172-261d-e301-985391782bd6",  "type":"",  "exec":"",  "is\_start":"0",  "description":"Saluta",  "x\_pos":"802",  "y\_pos":"300",  "last\_mod":"2021-01-19 15:58:34",  "elements":[  {  "elem\_id":"33ba6006-31fc-c51b-0331-88263260dc73",  "var\_name":"",  "e\_order":"1",  "note":"",  "d\_type":"LABEL",  "css":"",  "c\_type":"0",  "exit\_num":"",  "langs":{  "EN":{"label":"","uri":""},  "IT":{"label":"Ciao $nome","uri":""}  }  }  ],  "exits":[  {"exit\_dir":"0"},  {"exit\_dir":"0"}  ]  }  ]  }  ] |
| *Schema 5 – rappresentazione JSON di un intero workflow d’esempio* |

# Database di servizio a Flussu

# Database

Il database è diviso in tabelle che rispecchiano logicamente gli elementi che compongono il processo.

## Disegno logico delle tabelle del processo

Il disegno “logico” del database di Flussu è diviso in 3 macro aree, quella del processo in sé, quella della sessione d’uso e quello degli utenti che possono intervenire nel disegno, verifica e modifica dei workflow.

La numerazione delle tabelle definisce l’entità per il *workflow*, l’entità per i *blocchi*, in una relazione 1 a N col workflow, quella per le *uscite*, in relazione 1 a N col *blocco*.

L’entità per gli *elementi* prevede una relazione 0 a N col *blocco*, in quanto un *blocco* può anche non contenere alcun elemento (tipicamente lo start-block o il blocco di codice eseguibile)

L’entità blocco contiene le informazioni relative al *blocco* e ha un collegamento 1 a N con l’entità relativa al testo in lingua.

|  |
| --- |
| **T30 element**  1…N  **T40**  **language**  0…N  **T20 blocks**  1…N  **T25**  **block-->exit**  1…N  **T10**  **workflow** |
| *Fig.K– Disegno logico del database – PROCESSI/ENTITA’* |

# Processo

La tabella T10 definisce i dati di un processo

|  |
| --- |
| CREATE TABLE `**t10\_workflow**` (  `c10\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c10\_name` varchar(128) NOT NULL DEFAULT 'undefined',  `c10\_description` tinytext DEFAULT NULL,  `c10\_supp\_langs` varchar(128) NOT NULL DEFAULT 'EN',  `c10\_userid` int(10) unsigned NOT NULL,  `c10\_validfrom` datetime DEFAULT '1899-12-31 23:59:59',  `c10\_validuntil` datetime DEFAULT '1899-12-31 23:59:59',  `c10\_active` int(2) unsigned NOT NULL DEFAULT 1,  `c10\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c10\_modified` timestamp NOT NULL DEFAULT current\_timestamp(),  `c10\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c10\_deleted\_by` int(11) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c10\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T10\_workflow* |

### Backup copy

Contiene il disegno JSON “puro” quale backup automatico dell’ultimo salvataggio.

Contiene al massimo 10 salvataggi per ogni workflow.

|  |
| --- |
| CREATE TABLE `**t15\_workflow\_backup**` (  `c15\_backup\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c15\_workflow\_id` int(10) unsigned NOT NULL,  `c15\_workflow\_json` text COLLATE utf8mb4\_unicode\_ci NOT NULL,  `c15\_rec\_date` timestamp NOT NULL DEFAULT current\_timestamp(),  PRIMARY KEY (`c15\_backup\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci; |
| *Database - tabella T15\_workflow\_backup* |

## Blocco

La tabella T20 contiene i dati dei singoli blocchi

|  |
| --- |
| CREATE TABLE `**t20\_block**` (  `c20\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c20\_uuid` varchar(36) DEFAULT NULL,  `c20\_flofoid` int(10) unsigned NOT NULL,  `c20\_start` int(2) unsigned NOT NULL DEFAULT 0,  `c20\_desc` varchar(128) DEFAULT NULL,  `c20\_exec` mediumtext DEFAULT NULL,  `c20\_type` varchar(128) DEFAULT NULL,  `c20\_xpos` float DEFAULT 0,  `c20\_ypos` float DEFAULT 0,  `c20\_note` tinytext DEFAULT NULL,  `c20\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c20\_modified` timestamp NOT NULL DEFAULT current\_timestamp(),  `c20\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c20\_deleted\_by` int(11) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c20\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T20\_block* |

## Uscite dei blocchi

La tabella T25 contiene le informazioni sulle uscite

|  |
| --- |
| CREATE TABLE `**t25\_blockexit**` (  `c25\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c25\_blockid` int(10) unsigned NOT NULL,  `c25\_nexit` tinyint(3) unsigned NOT NULL DEFAULT 0,  `c25\_direction` int(10) unsigned NOT NULL DEFAULT 0,  `c25\_description` varchar(45) NOT NULL DEFAULT '',  `c25\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c25\_modified` timestamp NOT NULL DEFAULT current\_timestamp(),  `c25\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c25\_deleted\_by` int(11) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c25\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T20\_blockexit* |

## Elementi

La tabella T30 è una tabella che contiene le informazioni degli elementi e il puntatore ai record della tabella T40 per la parte relativa al testo in lingua

|  |
| --- |
| CREATE TABLE `**t30\_blk\_elm**` (  `c30\_elemid` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c30\_blockid` int(10) unsigned NOT NULL,  `c30\_uuid` varchar(36) DEFAULT NULL,  `c30\_varname` varchar(128) NOT NULL DEFAULT '',  `c30\_type` int(3) unsigned NOT NULL DEFAULT 0,  `c30\_exit\_num` int(10) unsigned DEFAULT NULL,  `c30\_css` varchar(45) DEFAULT NULL,  `c30\_order` int(3) unsigned NOT NULL DEFAULT 0,  `c30\_note` tinytext DEFAULT NULL,  `c30\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c30\_modified` timestamp NOT NULL DEFAULT current\_timestamp(),  `c30\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c30\_deleted\_by` int(11) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c30\_elemid`,`c30\_blockid`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T30\_blk\_elm* |

## Testo in lingua

La tabella T40 contiene le informazioni della lingua per ogni elemento (t30)

|  |
| --- |
| CREATE TABLE `**t40\_element**` (  `c40\_id` int(10) unsigned NOT NULL,  `c40\_lang` varchar(5) NOT NULL DEFAULT 'EN',  `c40\_text` mediumtext DEFAULT NULL,  `c40\_url` varchar(45) DEFAULT NULL,  `c40\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c40\_modified` timestamp NOT NULL DEFAULT current\_timestamp(),  `c40\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c40\_deleted\_by` int(11) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c40\_id`,`c40\_lang`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T40\_element* |

## Disegno logico delle tabelle degli utenti

Gli utenti del sistema sono autenticati con userid e password e hanno autorizzazioni che differiscono per ruolo (*vd*: Fig.J, tabella T90\_role).

|  |
| --- |
| **T90**  **role**  **T80**  **user**  1…1 |
| *Fig.L – Disegno logico del database – UTENTI/RUOLI* |

## Utenti del sistema

La tabella T80 contiene le informazioni relative agli utenti del sistema workflow

|  |
| --- |
| CREATE TABLE `**t80\_user**` (  `c80\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  `c80\_email` varchar(65) NOT NULL,  `c80\_username` varchar(65) NOT NULL,  `c80\_password` varchar(250) NOT NULL,  `c80\_pwd\_chng` datetime NOT NULL DEFAULT current\_timestamp(),  `c80\_role` int(4) unsigned DEFAULT 0,  `c80\_name` varchar(60) DEFAULT NULL,  `c80\_surname` varchar(60) DEFAULT NULL,  `c80\_created` timestamp NOT NULL DEFAULT current\_timestamp(),  `c80\_modified` timestamp NOT NULL DEFAULT current\_timestamp() ON UPDATE current\_timestamp(),  `c80\_deleted` datetime DEFAULT '1899-12-31 23:59:59',  `c80\_deleted\_by` int(10) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c80\_id`),  UNIQUE KEY `UNQ\_UserName` (`c80\_username`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T80\_user* |

## Ruoli degli utenti

La tabella T90 contiene le informazioni relative alle autorizzazioni degli utenti del sistema.

|  |
| --- |
| CREATE TABLE `**t90\_role**` (  `c90\_id` int(4) unsigned NOT NULL,  `c90\_name` varchar(30) NOT NULL,  `c90\_crud` varchar(5) NOT NULL DEFAULT 'CRUDX',  PRIMARY KEY (`c90\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T90\_role* |

I ruoli, predefiniti, alla v1.0 sono rappresentati nella seguente tabella:

|  |
| --- |
|  |
| *Fig.M – ruoli previsi nel sistema v1.0* |

# Esecuzione dei processi

Come affrontato nei capitoli precedenti, all’esecuzione di un processo viene generata un’area di memoria relativa alla sessione del processo. Quest’area, totalmente gestita dal motore Flussu, ha la durata standard di 24 ore, poi viene eliminata da un evento del database.

Prima dell’eliminazione vengono eseguite altre routines che generano sintesi dei dati d’accesso, specificata nel capitolo relativo ai “dati statistici”.

|  |
| --- |
| 1…N  N…1  **T200**  **worker**  **T205**  **work\_var**  **T209**  **work\_log** |
| *Fig.N – Disegno logico del database – SESSIONI DI LAVORO* |

## Dati della sessione del processo

La tabella T200 contiene le informazioni relative all’esecuzione di un workflow.

Il numero di processi in corso corrisponde al numero delle righe di questa tabella.

L’emivita di una sessione è definita in un EVENT del database, che si occupa ovviamente anche del log e delle variabili, è pari a 3 ore dall’ultima registrazione.

|  |
| --- |
| CREATE TABLE `**t200\_worker**` (  `c200\_sess\_id` varchar(36) NOT NULL,  `c200\_start` timestamp NOT NULL DEFAULT current\_timestamp(),  `c200\_wid` int(10) unsigned NOT NULL DEFAULT 0,  `c200\_lang` varchar(5) DEFAULT NULL,  `c200\_thisblock` varchar(36) NOT NULL DEFAULT '0',  `c200\_time\_start` datetime NOT NULL DEFAULT current\_timestamp(),  `c200\_state\_error` int(2) unsigned NOT NULL DEFAULT 0,  `c200\_state\_usererr` int(2) unsigned NOT NULL DEFAULT 0,  `c200\_state\_exterr` int(2) unsigned NOT NULL DEFAULT 0,  `c200\_blk\_end` int(10) unsigned DEFAULT NULL,  `c200\_time\_end` datetime NOT NULL DEFAULT current\_timestamp(),  `c200\_user` int(10) unsigned NOT NULL DEFAULT 0,  PRIMARY KEY (`c200\_sess\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T200\_worker* |

## Dati delle variabili della sessione

La sessione contiene le variabili generate dal processo durante la sua esecuzione. Questi dati sono registrati in una apposita tabella.

La tabella T205 contiene queste informazioni.

|  |
| --- |
| CREATE TABLE `**t205\_work\_var**` (  `c205\_sess\_id` varchar(36) NOT NULL,  `c205\_elm\_id` varchar(50) NOT NULL,  `c205\_elm\_val` mediumtext DEFAULT NULL,  `c205\_timestamp` timestamp NULL DEFAULT current\_timestamp(),  `c205\_source` int(4) unsigned NOT NULL DEFAULT 0 COMMENT '0=user\n1=Flussu\n2=ext\_svc',  PRIMARY KEY (`c205\_sess\_id`,`c205\_elm\_id`) USING BTREE  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T205\_work\_var* |

## Dati del log della sessione

La tabella T209 contiene le informazioni del registro interno delle operazioni, utile ad eventuale debug

|  |
| --- |
| CREATE TABLE `**t209\_work\_log**` (  `c209\_sess\_id` varchar(36) NOT NULL,  `c209\_timestamp` timestamp NOT NULL DEFAULT current\_timestamp(),  `c209\_tpinfo` int(4) unsigned NOT NULL DEFAULT,  `c209\_row` mediumtext DEFAULT NULL,  KEY `ix\_t200\_log` (`c209\_sess\_id`,`c209\_timestamp`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |
| *Database - tabella T209\_work\_log* |

# Dati statistici

Usando gli eventi del motore di database, ogni ora vengono sintetizzati i dati d’uso in dati statistici, che poi vengono archiviati in modo da permettere di generare grafici d’uso in unità di tempo.

Oltra a ciò vengono calcolati i dati di accesso dalle diverse piattaforme, con ciò intendendo accesso da web o da chat/ app come Telegram, Whastapp, FB/Messenger, ecc.

|  |
| --- |
| **T70**  **stat**  **T71**  **access** |
| *Fig.O – Disegno logico del database – STATISTICA* |

## Dati delle sessioni d’uso

La tabella **t70\_stat** contiene le informazioni delle sessioni d’uso

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Colonna | Tipo | Null | Predefinito | Commenti |
| c70\_wid | int(10) | No |  |  |
| c70\_sid | varchar(36) | No |  |  |
| c70\_bid | int(10) | No |  |  |
| c70\_start | smallint(6) | No | 0 |  |
| c70\_channel | int(2) | No |  |  |
| c70\_timestamp | datetime | No | current\_timestamp() |  |
| c70\_data | mediumtext | No |  |  |
| c70\_tag | varchar(2) | No |  | internal use |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Chiave | Tipo | Unica | Compresso | Colonna | Codif. Car. | Null |
| ix\_wid | BTREE | No | No | c70\_wid | A | No |
| ix\_wid\_sid | BTREE | No | No | c70\_wid | A | No |
| c70\_sid | A | No |
| ix\_exec | BTREE | No | No | c70\_timestamp | A | No |
| ix\_start | BTREE | No | No | c70\_wid | A | No |
| c70\_sid | A | No |
| c70\_start | A | No |

## Canali d’accesso

La tabella **t71\_access** contiene le informazioni relative ai canali d’accesso d’uso

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Colonna | Tipo | Null | Predefinito | Commenti |
| c71\_wid *(Primaria)* | int(10) | No |  |  |
| c71\_sid *(Primaria)* | varchar(36) | No |  |  |
| c71\_date | datetime | No |  |  |
| c71\_chan | tinyint(4) | No |  |  |
| c71\_oth | varchar(10) | No |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Chiave | Tipo | Unica | Compresso | Colonna | Codifica car. | Null |
| PRIMARY | BTREE | Sì | No | c71\_wid | A | No |
| c71\_sid | A | No |
| ix\_71\_date | BTREE | No | No | c71\_date | A | No |
| ix\_71\_chan | BTREE | No | No | c71\_chan | A | No |

1. https://it.wikipedia.org/wiki/Diagramma\_di\_flusso [↑](#footnote-ref-1)
2. *vd*: https://emojipedia.org/ [↑](#footnote-ref-2)
3. *cfr:* https://www.html.it/pag/16679/le-variabili3/ [↑](#footnote-ref-3)
4. *cfr:* https://it.wikipedia.org/wiki/Sandbox\_(sicurezza\_informatica) [↑](#footnote-ref-4)
5. *cfr:* https://it.wikipedia.org/wiki/Spam [↑](#footnote-ref-5)
6. *cfr*: https://en.wikipedia.org/wiki/Plain\_text [↑](#footnote-ref-6)
7. *cfr*: https://it.wikipedia.org/wiki/One-time\_password [↑](#footnote-ref-7)
8. *cfr*: https://www.html.it/pag/62463/le-richieste-http-get-e-post/ [↑](#footnote-ref-8)
9. *cfr*: https://it.wikipedia.org/wiki/Programmazione\_orientata\_agli\_oggetti [↑](#footnote-ref-9)