JE - ADEnext

Costruire schermate, tabelle e flussi disegnandoli.

Cosa ci guadagni. Una funzione nuova si disegna in ore invece di programmarla in giorni, e resta modificabile.

- JE1 - A cosa serve
  - JE1.1 - Disegnare invece di programmare
  - JE1.2 - Cosa si può fare e cosa no
  - JE1.3 - Dove sta nella mappa
- JE2 - I dati
  - JE2.1 - Il catalogo di record e campi
  - JE2.2 - Elenchi chiusi
  - JE2.3 - Regole di cancellazione
  - JE2.4 - Valori dell'installazione
- JE3 - I form
  - JE3.1 - Disegnare un form
  - JE3.2 - Regole e controlli
  - JE3.3 - Scrivere valori e formule
- JE4 - Le tabelle
  - JE4.1 - Costruire una tabella
  - JE4.2 - Filtri, gruppi e totali
  - JE4.3 - Stili e azioni
- JE5 - Gli endpoint
  - JE5.1 - Leggere dati senza scrivere codice
  - JE5.2 - Raggruppare e contare
  - JE5.3 - Scritture disegnate

- JE6 - I flussi
  - JE6.1 - Il primo flusso
  - JE6.2 - Nodi e proprietà
  - JE6.3 - Partire da solo: i trigger
  - JE6.4 - Fermarsi e riprendere
  - JE6.5 - Versioni e ripristino
  - JE6.6 - Trovare gli errori
- JE7 - Viste e variabili
  - JE7.1 - Mettere insieme form, tabelle e grafici
  - JE7.2 - Le variabili dei componenti
  - JE7.3 - Se qualcosa non va

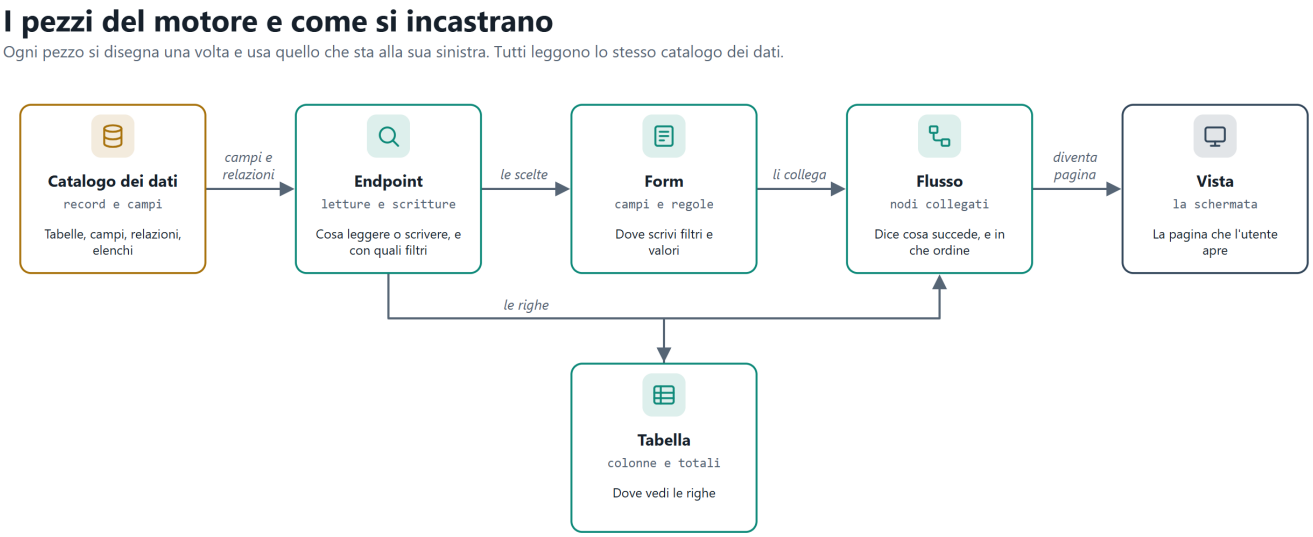
JE1 - A cosa serve

JE1.1 - Disegnare invece di programmare

In una frase. ADEnext è il motore con cui una funzione del gestionale si disegna a schermo, pezzo per pezzo, invece di scriverla come programma.

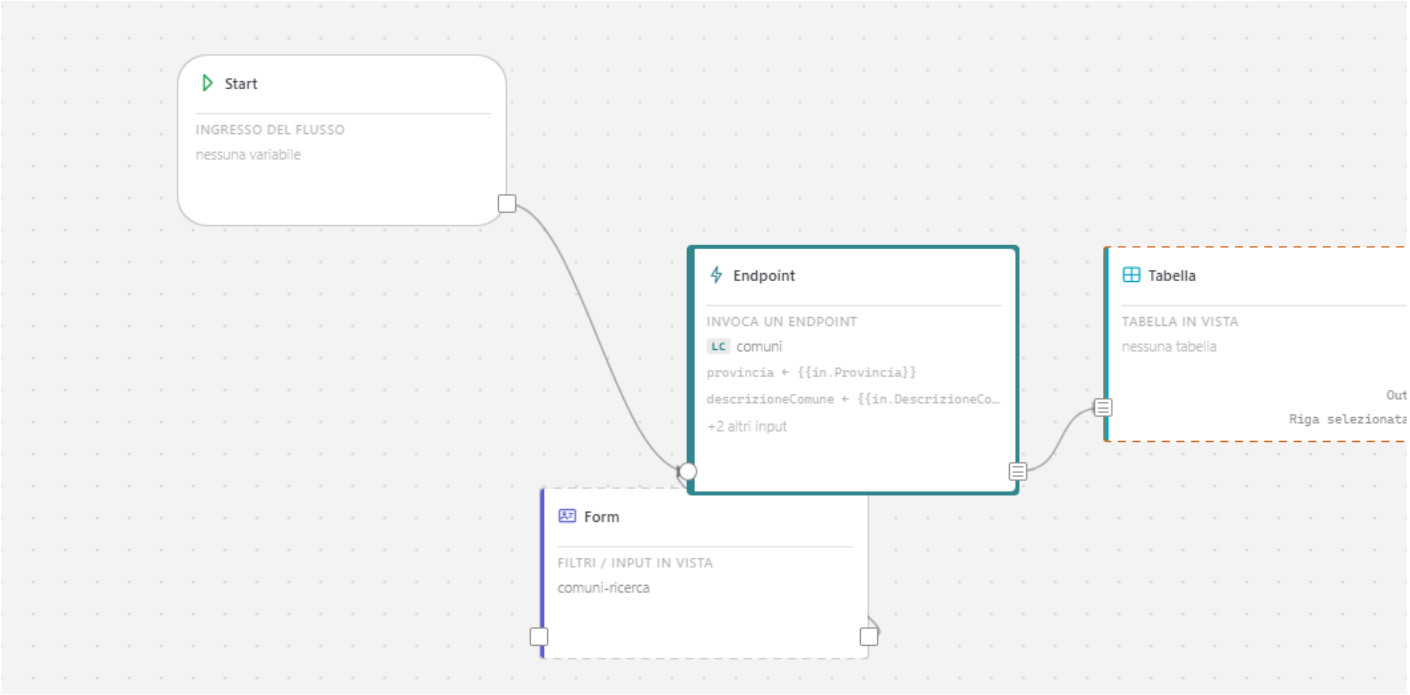
Una funzione del gestionale è quasi sempre fatta delle stesse cose: dei dati da leggere, una maschera dove scrivere i filtri, una tabella con i risultati, qualche regola. Scriverle ogni volta a mano vuol dire giorni di lavoro di un programmatore, e altri giorni a ogni modifica.

Con ADEnext quei pezzi si disegnano. Scegli i campi da un elenco, trascini, colleghi, salvi. Il disegno resta scritto in un archivio, e la pagina che l'utente apre lo rilegge ogni volta: se cambi il disegno, cambia la pagina.

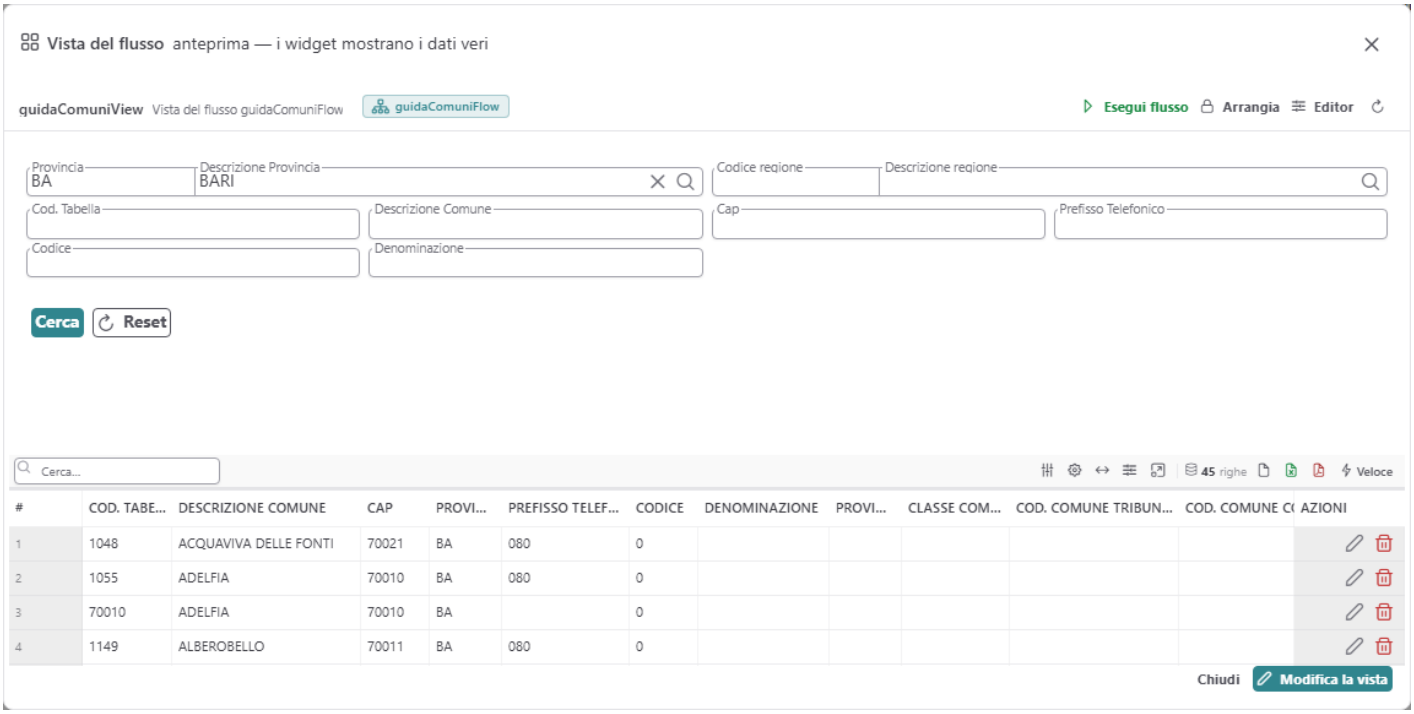


I pezzi del motore: dal catalogo dei dati alla pagina che l'utente apre

Qui sotto un esempio vero, la ricerca dei comuni. Prima il disegno: quattro nodi collegati. Poi la pagina che ne esce, con i filtri sopra e la tabella sotto.



Il disegno: all'apertura e a ogni ricerca, l'endpoint legge i comuni e li passa alla tabella



La pagina che ne esce: scrivi la provincia, premi Cerca, la tabella mostra i 45 comuni trovati

Cosa ci guadagni.

• **Una funzione nuova si disegna, non si programma.** Una ricerca con filtri, tabella e azioni nasce in sei passi guidati. Nessuno scrive codice, nessuno ricompila.

• **Resta modificabile.** Aggiungere una colonna, cambiare un controllo, spostare un campo: riapri il disegno, cambi, salvi. Ogni salvataggio che cambia qualcosa lascia una versione, e si può tornare indietro.

• **Le regole si scrivono una volta.** Form, tabelle, endpoint e flussi leggono lo stesso

> catalogo dei dati. Un controllo scritto sul form vale a schermo e vale anche sul server, con lo stesso messaggio.

Questo libro segue l'ordine in cui i pezzi si incastrano: i dati, i form, le tabelle, gli endpoint, i flussi, le viste. Come il disegno diventa pagina è in Dove sta nella mappa.

> **Nota.** L'analisi dei dati con numeri e grafici ha il suo libro: JF - Analytics. Le stampe in PDF anche: JK - GPEnext.

> **Per chi sviluppa.** Il servizio è `backend/middleware/it_gsba_ade_next`. Nell'app web il motore è `frontend/ade-next/src`, in tre parti: `kernel` (modelli, regole, espressioni), `runtime` (form, tabelle, viste a schermo), `studio` (gli strumenti per disegnare).

JE1.2 - Cosa si può fare e cosa no

In una frase. Cosa ADEnext sa fare bene oggi, cosa fa solo in parte e cosa va ancora scritto a mano.

Prima di disegnare una funzione conviene sapere se il motore ci arriva. C'è un documento che lo dice riga per riga, e viene aggiornato ogni volta che il motore impara una cosa nuova. Questa pagina ne è il riassunto: per decidere davvero si guarda quello.

Si può, ed è solido

- **Leggere dati**: scegliere i campi, filtrare con «e» e «oppure», ordinare, unire archivi collegati, contare e sommare per gruppi, tenere solo i gruppi che superano una soglia.
- **Form**: una ventina di tipi di campo, griglia a dodici colonne, riquadri, schede e passi, finestre di ricerca per scegliere un valore collegato.
- **Regole**: campi obbligatori, visibili o bloccati a condizione, controlli fra due campi. Valgono a schermo e sul server.
- **Tabelle**: tipi di colonna, colonne calcolate, filtri, gruppi, totali, colori a condizione, azioni di riga, esportazione.
- **Versioni**: form, tabelle, endpoint e flussi tengono la storia dei salvataggi.

Si può, ma è giovane

- **Flussi**: condizioni, cicli, attese, attività affidate a una persona, avvio a orario. Funzionano e hanno i loro test. Molte parti non sono ancora state usate su dati veri di un cliente.
- **Scritture disegnate**: inserire, modificare e cancellare senza scrivere SQL. Restano spente finché l'installazione non le accende, archivio per archivio.
- **Grafici nelle viste**: cinque tipi (linee, barre, torta, ciambella, area).

Non si può, per ora

- Domande annidate una dentro l'altra, se non scrivendo l'SQL a mano.
- Più scritture diverse dentro un solo endpoint. Si mettono in fila in un flusso.
- Un contenitore a schede dentro una vista.

- Piccoli programmi scritti a mano dentro una vista per trasformare i dati.
- Generare in un colpo solo il codice di molti archivi.

> **Attenzione.** Se la funzione che ti serve cade nel terzo elenco, non si forza il motore: quella parte si programma. Tutto il resto può restare disegnato.

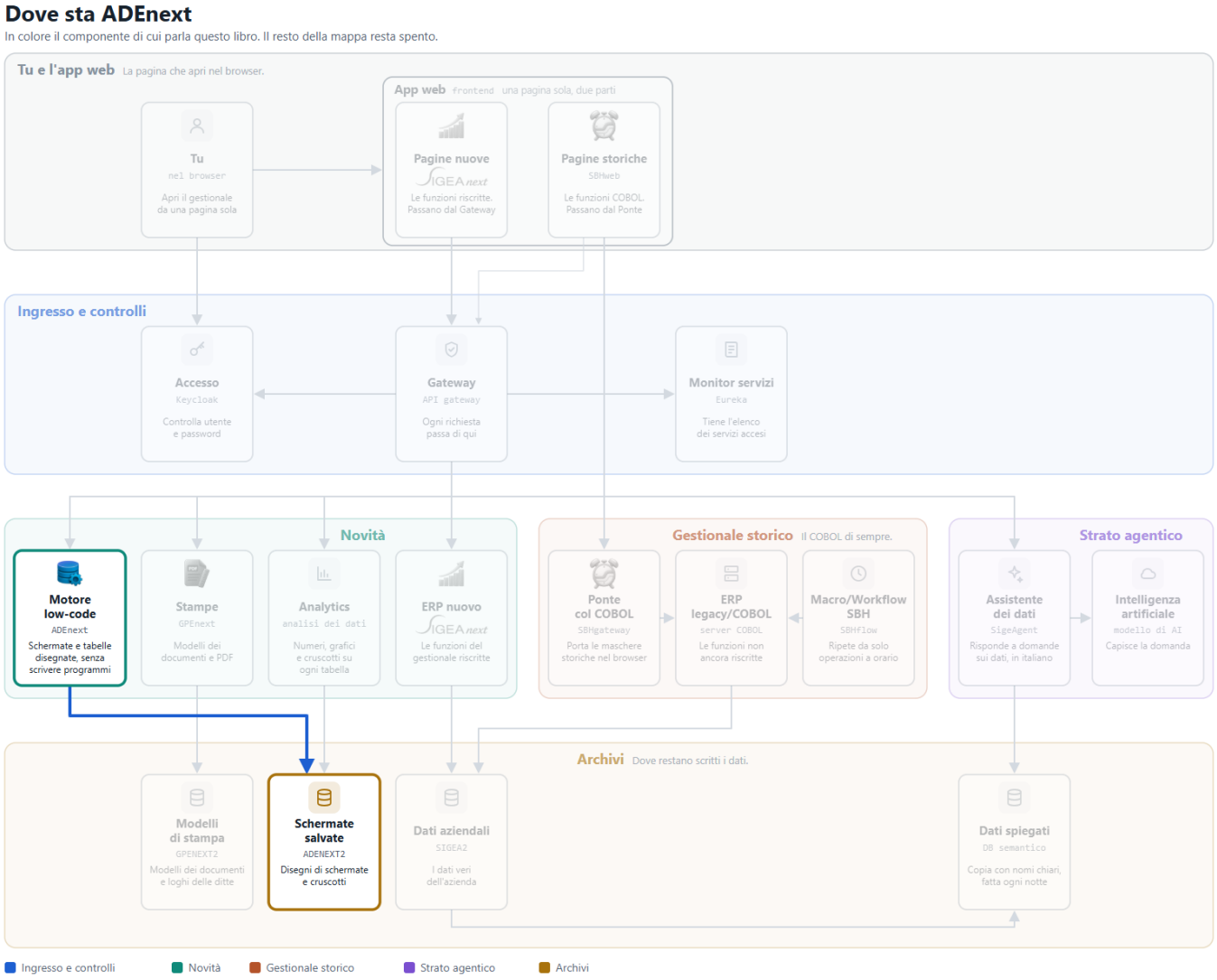
> **Per chi sviluppa.** Nel documento ogni riga ha uno stato (`SUPPORTATO`, `PARZIALE`, `NO`) e il file che lo prova. Le diciture come «non eseguito su dati veri» vanno prese alla lettera.

> **Per approfondire.**
> • Capabilities del motore low-code — stato reale al 2026-11-14: `docs/low-code/capabilities.md`

JE1.3 - Dove sta nella mappa

In una frase. ADEnext è uno dei servizi delle Novità: sta dietro il Gateway e tiene i disegni in un archivio suo, le Schermate salvate.

Sapere dove sta aiuta a capire due cose: dove finisce quello che disegni, e perché una pagina disegnata mostra gli stessi dati di ogni altra pagina del gestionale.



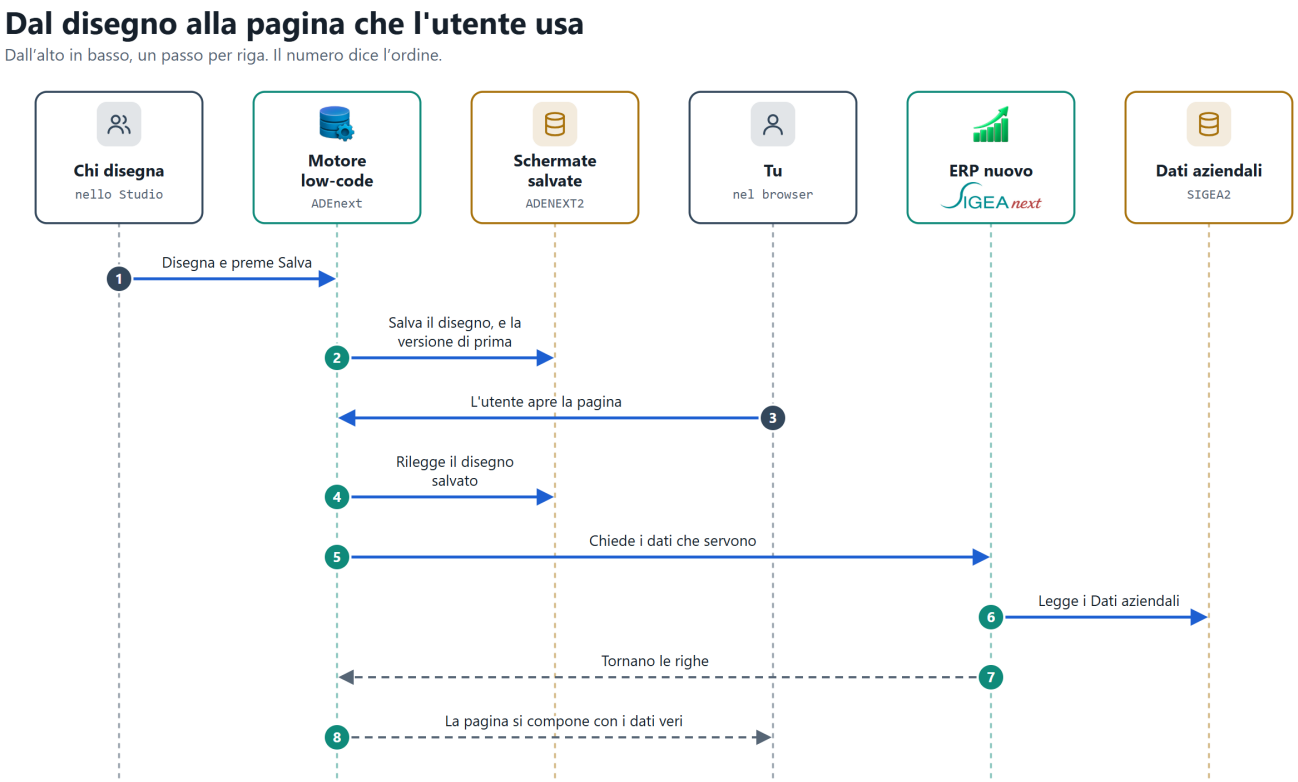
La mappa di ADSWnext con accesi ADEnext e il suo archivio, le Schermate salvate

Chi lo chiama. Le pagine nuove, ogni volta che aprono una schermata disegnata: chiedono ad ADEnext com'è fatta. Lo chiamano anche gli strumenti per disegnare, quando salvi. La richiesta passa dal Gateway, come tutte.

Chi chiama lui. Il suo archivio, **Schermate salvate**: lì stanno i disegni e le loro versioni. E SIGEAnext, quando servono i dati: ADEnext decide cosa chiedere, SIGEAnext legge i Dati aziendali

e rimanda le righe.

Cosa vuol dire per te. I disegni e i dati dell'azienda stanno in due archivi diversi. Cambiare un disegno non tocca un dato. E una pagina disegnata legge i dati dalla stessa strada delle altre: non ne esiste una copia.



Dal disegno alla pagina: chi disegna salva una volta, la pagina rilegge il disegno a ogni apertura

> **Nota.** Per usare una pagina disegnata basta il tuo accesso di sempre. Per disegnare serve il livello Utente avanzato. I livelli sono spiegati nel libro JB - Accesso e permessi.

> **Per chi sviluppa.**
> • Il Gateway manda ad ADEnext le richieste che cominciano per `/api/ade`. Il servizio ascolta sulla porta 5080.
> • L'archivio sul server di prova si chiama ADENEXT2.
> • Le letture e le scritture disegnate le esegue SIGEAnext (`backend/business-logic/it_gsba_sigea_next`). I record che arrivano dal catalogo dell'Assistente dei dati li esegue lui.
> • Una vista pubblicata si apre per nome, all'indirizzo `/ade-next/app/<nome>`.

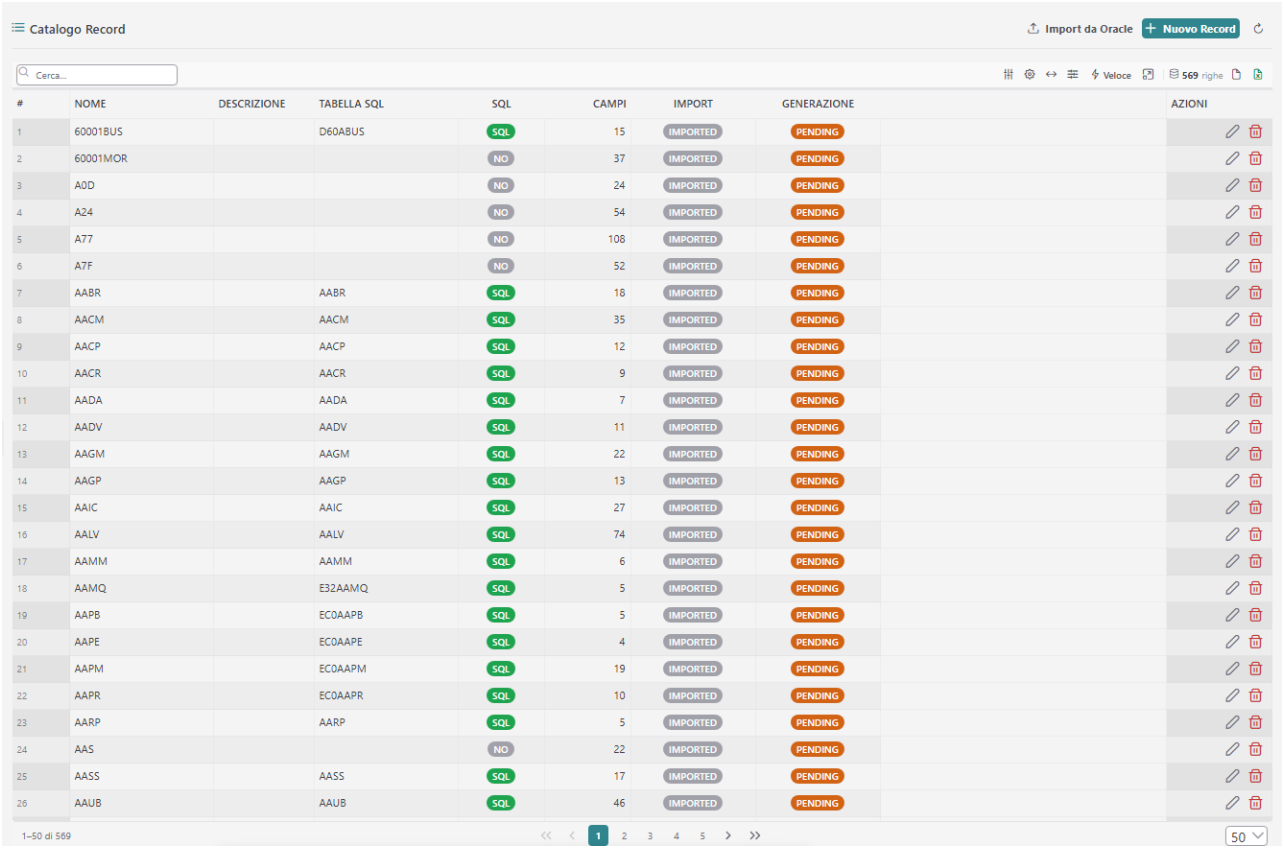
JE2 - I dati

JE2.1 - Il catalogo di record e campi

In una frase. Il catalogo è l'elenco di tutti gli archivi che il sistema conosce, con i loro campi: è da qui che form, tabelle ed endpoint prendono i nomi.

Tutto quello che disegni parte dai dati. Se ogni schermata si inventasse i nomi dei campi, due schermate sullo stesso archivio direbbero cose diverse. Il catalogo evita questo: un archivio (qui si chiama **record**) si descrive una volta, e tutti lo leggono da lì.

1. **Apri il catalogo.** Dal menu: **Strumenti di Sviluppo › Record & Tabelle › Catalogo Record**. Ogni riga è un archivio: nome, tabella, quanti campi ha, a che punto è.

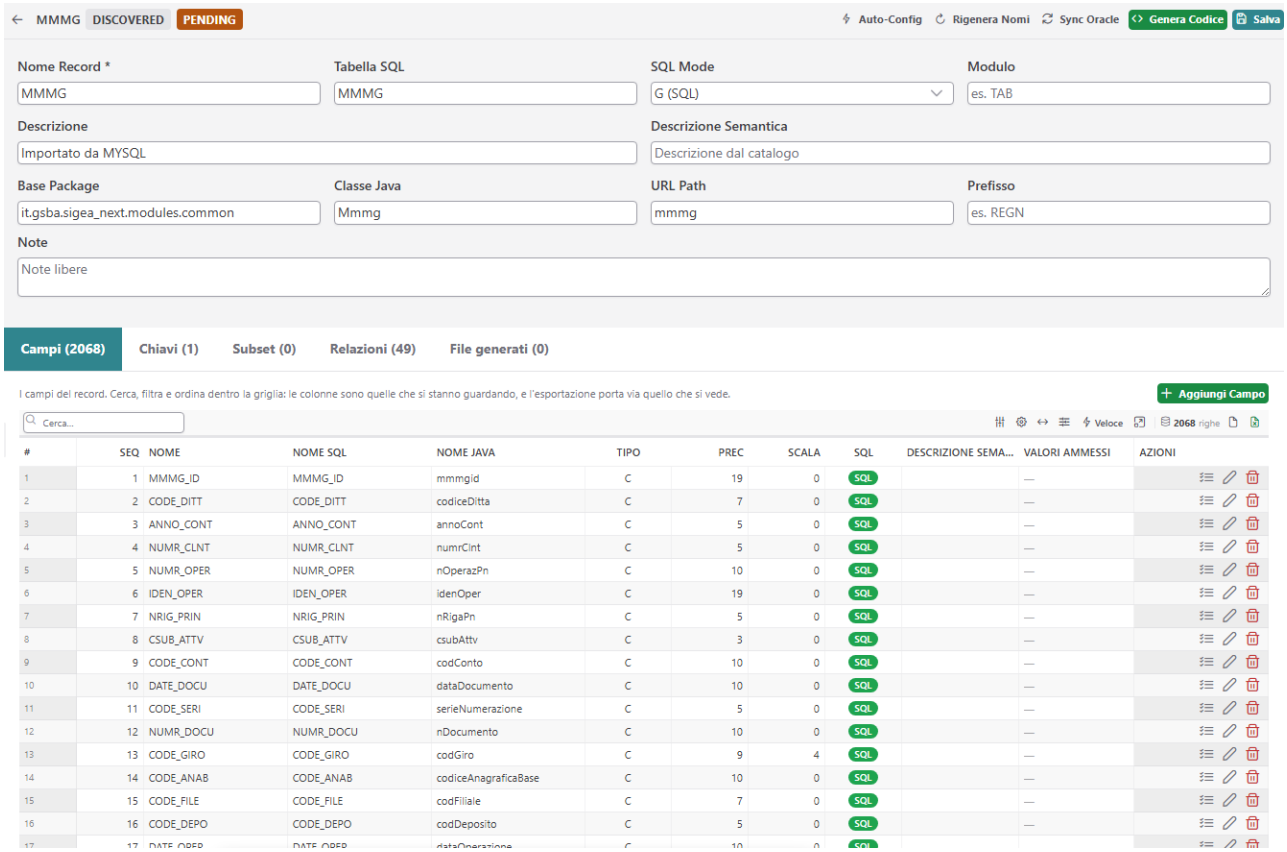


#	NOME	DESCRIZIONE	TABELLA SQL	SQL	CAMPI	IMPORT	GENERAZIONE	AZIONI
1	60001BUS		D60ABUS	SQL	15	IMPORTED	PENDING	
2	60001MOR			NO	37	IMPORTED	PENDING	
3	A0D			NO	24	IMPORTED	PENDING	
4	A24			NO	54	IMPORTED	PENDING	
5	A77			NO	108	IMPORTED	PENDING	
6	A7F			NO	52	IMPORTED	PENDING	
7	AABR		AABR	SQL	18	IMPORTED	PENDING	
8	AACM		AACM	SQL	35	IMPORTED	PENDING	
9	AACP		AACP	SQL	12	IMPORTED	PENDING	
10	AACR		AACR	SQL	9	IMPORTED	PENDING	
11	AADA		AADA	SQL	7	IMPORTED	PENDING	
12	AADV		AADV	SQL	11	IMPORTED	PENDING	
13	AAGM		AAGM	SQL	22	IMPORTED	PENDING	
14	AAGP		AAGP	SQL	13	IMPORTED	PENDING	
15	AAIC		AAIC	SQL	27	IMPORTED	PENDING	
16	AALV		AALV	SQL	74	IMPORTED	PENDING	
17	AAMM		AAMM	SQL	6	IMPORTED	PENDING	
18	AAMQ		E32AAMQ	SQL	5	IMPORTED	PENDING	
19	AAPB		EC0AAPB	SQL	5	IMPORTED	PENDING	
20	AAPE		EC0AAPE	SQL	4	IMPORTED	PENDING	
21	AAPM		EC0AAPM	SQL	19	IMPORTED	PENDING	
22	AAPR		EC0AAPR	SQL	10	IMPORTED	PENDING	
23	AARP		AARP	SQL	5	IMPORTED	PENDING	
24	AAS			NO	22	IMPORTED	PENDING	
25	AASS		AASS	SQL	17	IMPORTED	PENDING	
26	AAUB		AAUB	SQL	46	IMPORTED	PENDING	

1–50 di 569 — 1 2 3 4 5 — 50

Il Catalogo Record: 569 archivi, con la ricerca e i filtri su ogni colonna

2. **Cerca.** Scrivi nella casella **Cerca** o usa il filtro sull'intestazione di una colonna. La ricerca guarda tutto il catalogo, non solo la pagina che hai davanti. Il numero in alto dice quanti ne hai trovati.
3. **Apri un record.** Doppio clic sulla riga. In alto i dati dell'archivio, sotto le schede: **Campi**, **Chiavi**, **Relazioni**.



Un record aperto: i suoi campi, con nome, tipo e lunghezza

4. **Guarda le relazioni.** La scheda **Relazioni** dice a quali altri archivi è collegato questo. Sono queste a permettere, più avanti, di aprire le righe di un ordine dalla sua testata, o di mostrare la descrizione accanto a un codice.

Il **Catalogo Campi**, nella stessa voce di menu, fa la stessa cosa al contrario: parte da un campo e ti dice in quali archivi compare.

> **Cosa ci guadagni.** Un campo ha un nome solo in tutto il sistema. Quando disegni non lo scrivi: lo scegli da un elenco, e non puoi sbagliarlo.

> **Per chi sviluppa.** Il catalogo è il registro dei dati di ADEnext. Si riempie importando la struttura dal database o dai tracciati dell'ERP legacy/COBOL. Le scritture disegnate sono ammesse solo sugli archivi che sono nel registro.

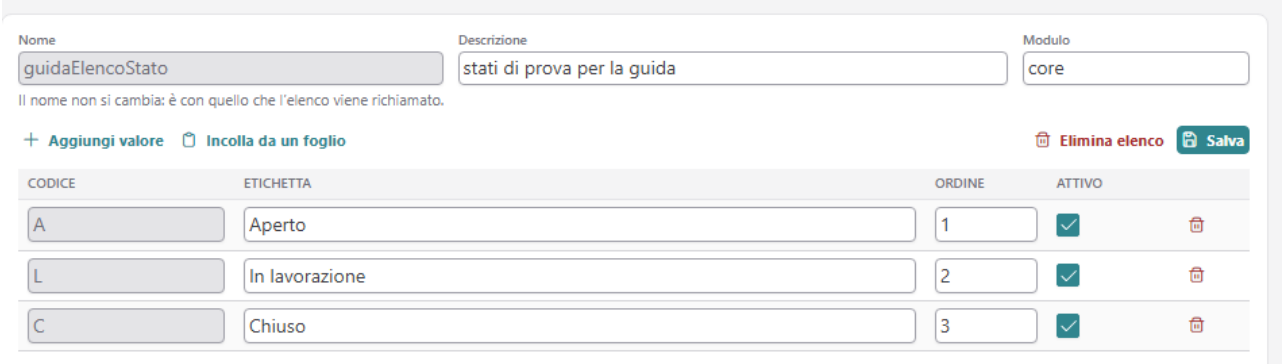
> **Per approfondire.**
> • Catalogo record e campi — guida passo passo: `docs/guide/dati/catalogo-record/README.md`

JE2.2 - Elenchi chiusi

In una frase. Un elenco chiuso è un insieme di valori ammessi, come gli stati di un ordine: lo scrivi una volta e lo usano tendine, tabelle, filtri e controlli.

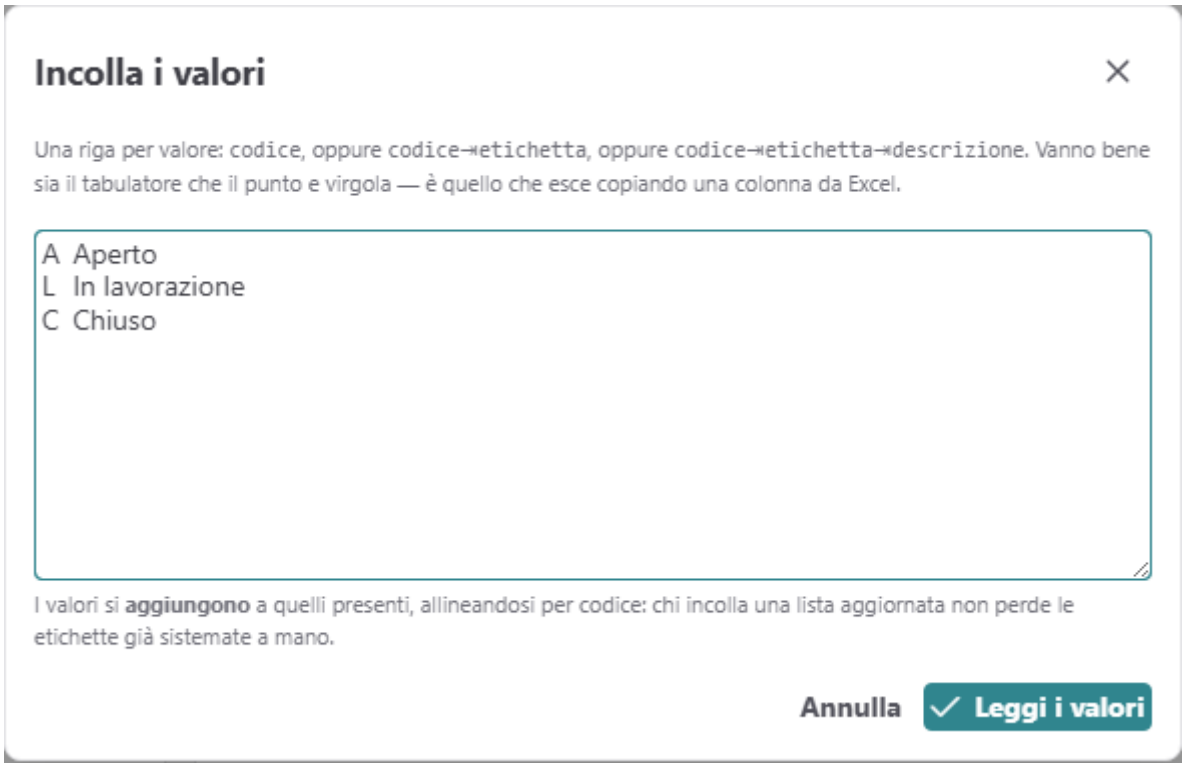
Senza un elenco in comune, gli stessi tre stati finiscono ricopiati in ogni campo che li usa, e prima o poi le copie non sono più uguali. Con l'elenco chiuso c'è un posto solo. E il sistema può controllare che il valore scritto sia uno di quelli ammessi.

1. **Apri la pagina.** Dal menu: **Strumenti di Sviluppo › Record & Tabelle › Elenchi chiusi**. Con il **+** crei un elenco nuovo.
2. **Scrivi i valori.** Ogni valore ha un **codice** e un'**etichetta**. Il codice è quello che resta scritto nei dati. L'etichetta è quello che legge la persona.



Un elenco salvato: tre stati, ciascuno con codice, etichetta e ordine. Elenco di prova

3. **Oppure incollali.** Se l'elenco esiste già in un foglio di calcolo, premi **Incolla da un foglio**: una riga per valore, codice ed etichetta. Per gli elenchi lunghi c'è **Carica un foglio**.



Incolla da un foglio: una riga per valore, il codice e poi l'etichetta

4. **Salva.** Da questo momento il codice è bloccato: cambiarlo lascerebbe senza etichetta i dati già scritti. L'etichetta si può cambiare quando vuoi.

> **Nota.** Un valore non si cancella, si **disattiva**. Non si può più scegliere, ma i dati di ieri che lo contengono continuano a mostrare la sua etichetta.

> **Cosa ci guadagni.** Un valore fuori elenco viene rifiutato, anche se arriva da un altro programma e non dalla pagina. Un campo lasciato vuoto non è un errore: quello lo decide la regola «obbligatorio».

> **Per chi sviluppa.** Nelle regole di un form il controllo è di tipo `staticEntity`, con il nome dell'elenco. Gli strumenti sono in `frontend/ade-next/src/studio/static-entity`.

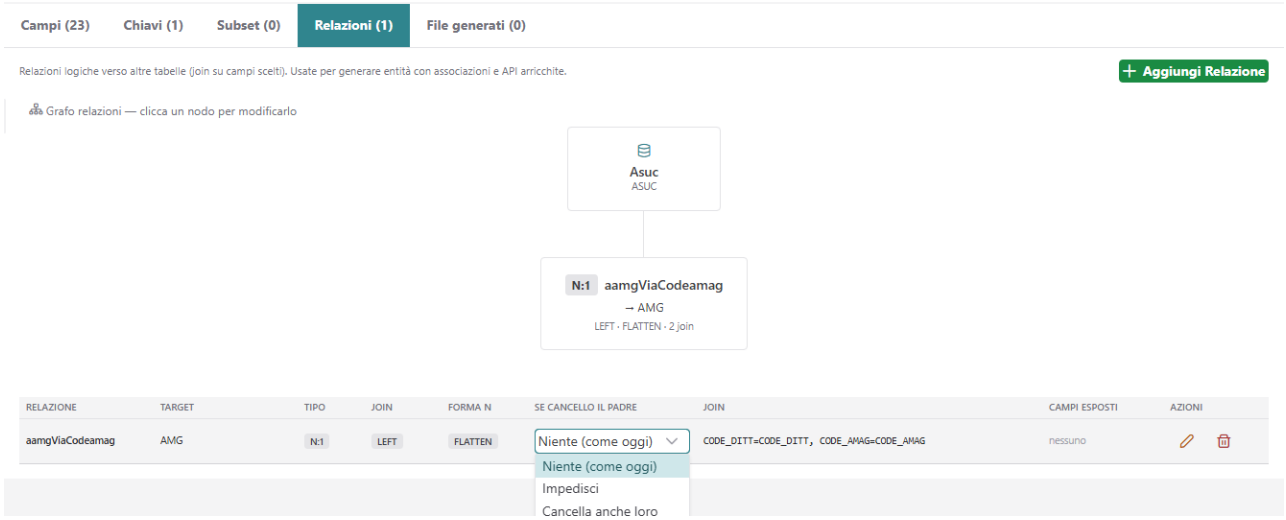
> **Per approfondire.**
> • Elenchi chiusi (static entity): `docs/guide/dati/elenchi-chiusi/README.md`

JE2.3 - Regole di cancellazione

In una frase. Per ogni collegamento fra due archivi decidi cosa succede alle righe collegate quando cancelli quella da cui dipendono.

Pensa a un ordine con le sue righe. Se cancelli l'ordine e nessuno si occupa delle righe, restano lì a puntare a un ordine che non esiste più. Nessun errore, nessun avviso: lo si scopre mesi dopo. La regola di cancellazione serve a deciderlo prima.

1. **Apri il record.** Dal **Catalogo Record**, doppio clic sull'archivio, poi scheda **Relazioni**.
2. **Scegli la regola.** Nella colonna **Se cancello il padre** ogni collegamento ha una tendina con tre risposte.



La scheda Relazioni: per ogni collegamento, la tendina «Se cancello il padre»

3. **Leggi cosa fanno.**
   - **Niente (come oggi)**: non cambia nulla. È la scelta di partenza, così nessuna cancellazione che già esiste cambia comportamento.
   - **Impedisci**: se ci sono righe collegate, la cancellazione non parte, e ti dice quante sono.
   - **Cancella anche loro**: le righe collegate vengono cancellate insieme. O spariscono tutte, o non sparisce niente.

La scelta si salva subito. Se il salvataggio non riesce, la tendina torna al valore di prima.

> **Cosa ci guadagni.** La regola vale ovunque: da una pagina, da un flusso, da un altro programma. L'anteprima di una cancellazione mostra anche quello che la regola aggiunge.

> **Attenzione.** Le regole si applicano alle cancellazioni disegnate. Hanno i loro test, ma non sono ancora state usate su dati veri: accendile una relazione per volta, e prova prima

sull'ambiente di prova.

Per chi sviluppa. Nel database i tre valori sono `IGNORE`, `PROTECT`, `DELETE`, sulla relazione del record.

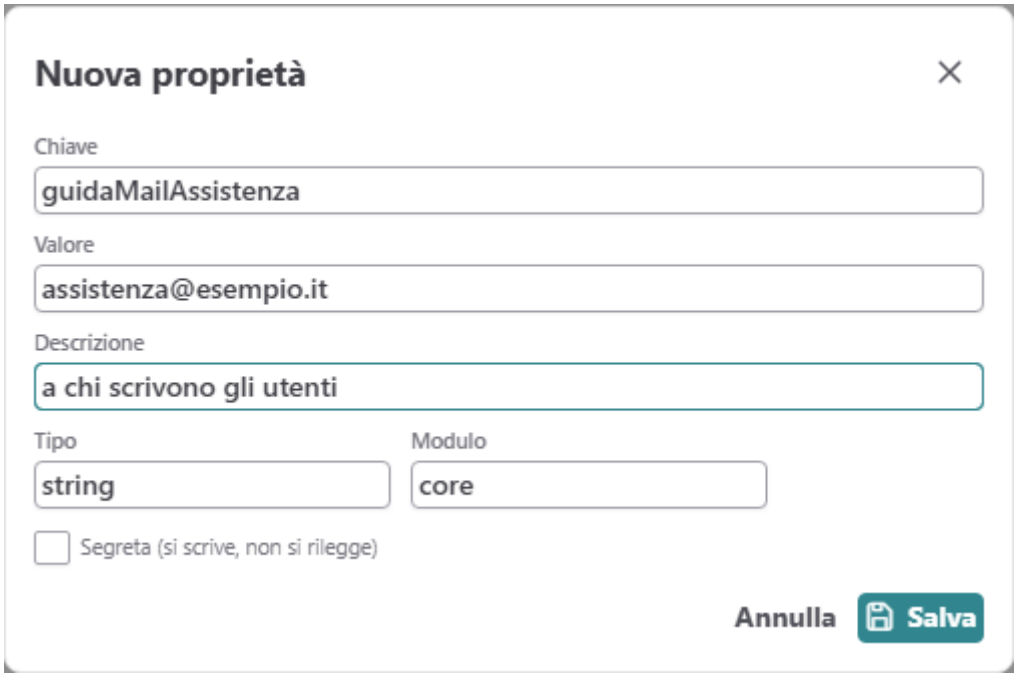
Per approfondire.
• Regole di cancellazione: `docs/guide/dati/regole-cancellazione/README.md`

JE2.4 - Valori dell'installazione

In una frase. Qui stanno i valori che cambiano da un'installazione all'altra, come la mail dell'assistenza, e i contatori che danno il numero ai documenti.

Certe cose non sono né dati né programma: l'indirizzo a cui scrivono gli utenti, la soglia oltre cui un ordine va approvato. Se finiscono scritte dentro una schermata, per cambiarle bisogna cercarle una per una. Qui hanno un posto solo, e ogni installazione ha i suoi.

1. **Apri la pagina.** Dal menu: **Strumenti di Sviluppo › Record & Tabelle › Valori installazione**. Ci sono due schede: **Proprietà** e **Numerazioni**.
2. **Aggiungi una proprietà.** Una chiave, un valore, una descrizione. Nei flussi la leggi con il suo nome.



Una proprietà nuova: chiave, valore e a cosa serve. Dati di prova

3. **Se è una password, spunta Segreta.** Da quel momento il valore si può scrivere ma non rileggere: l'elenco dice solo se è impostato. Salvare la descrizione senza riscrivere il valore non lo cancella.
4. **Aggiungi una numerazione.** Un nome e un formato. Nel formato `%y` è l'anno e `%06s` è il contatore su sei cifre: `FT-%y-%06s` dà `FT-2026-000001`.



Una numerazione nuova: il formato decide come esce il numero

Cosa ci guadagni. Due persone che salvano nello stesso istante non ricevono mai lo stesso numero. Con il vecchio «prendo l'ultimo e aggiungo uno» poteva succedere.

Attenzione. Un numero preso è preso, anche se poi il documento non viene salvato: la numerazione può avere buchi. Il contatore non si riporta indietro.

Per chi sviluppa. In un flusso una proprietà si legge come `{{$site.chiave}}`, una volta per esecuzione. Gli strumenti sono in `frontend/ade-next/src/studio/site-property`.

Per approfondire.
• Valori dell'installazione e numerazioni: `docs/guide/dati/valori-installazione/README.md`

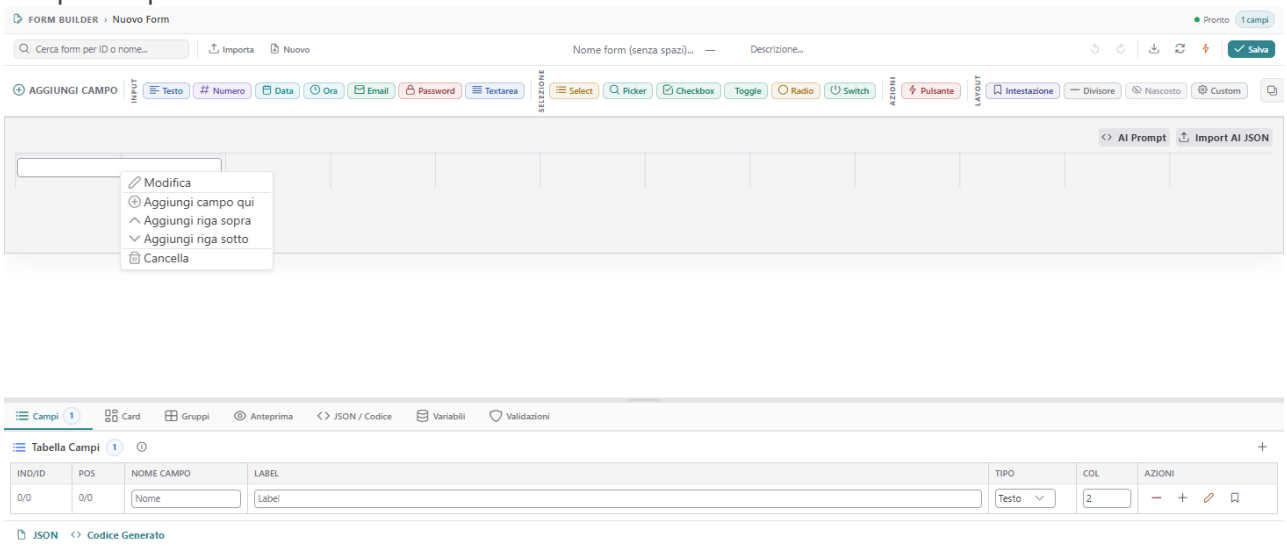
JE3 - I form

JE3.1 - Disegnare un form

In una frase. Un form è la maschera dove l'utente scrive: la componi scegliendo i campi e mettendoli su una griglia, e la pagina vera esce uguale a come l'hai disegnata.

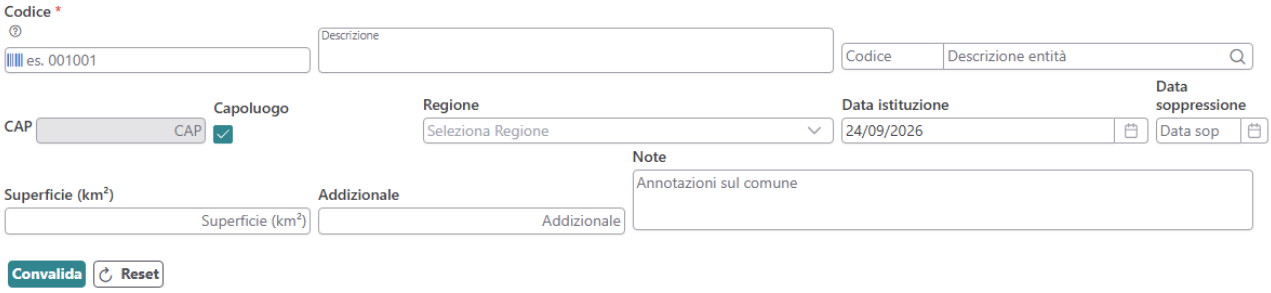
Un form disegnato non è un bozzetto da passare a un programmatore. È già la maschera che l'utente userà. Per questo il risultato si vede subito, e una modifica costa pochi minuti.

1. **Apri il Form Builder.** Dal menu: **Low-Code Platform › Dynamic Forms › Form Builder**. Puoi partire da un modello pronto (persona, azienda, veicolo, filtro di ricerca) o **Da zero**.
2. **Aggiungi i campi.** La barra in alto ha i tipi di campo, in quattro famiglie: testo, numero, data, scelta da un elenco, casella da spuntare, pulsante, titolo e altri. Un clic e il campo va nel primo posto libero.



Il Form Builder: in alto i tipi di campo, al centro la griglia, sotto l'elenco dei campi

3. **Mettili in ordine.** La griglia ha dodici colonne. Trascini un campo per spostarlo, tiri l'angolo per allargarlo. Con il tasto destro aggiungi una riga sopra o sotto. Ogni gesto si annulla con **Annulla**.
4. **Dai un nome a ogni campo.** Doppio clic sul campo: si apre il pannello a destra. Lì scrivi nome, etichetta, se è obbligatorio, il testo di aiuto, il valore con cui nasce.
5. **Raggruppa, se il form è lungo.** I campi si chiudono in riquadri con un titolo, oppure in schede o in passi con **Precedente** e **Successivo**.
6. **Guarda l'anteprima e salva.** La scheda **Anteprima** mostra il form vero. **Genera dati di test** lo riempie per provarlo. Poi **Salva**.



Lo stesso form nella pagina: campi, asterischi e aiuti sono quelli scelti nel disegno

Cosa ci guadagni. Per scegliere un valore collegato, come una provincia, c'è un tipo di campo già pronto: una finestra di ricerca con codice e descrizione. Non va costruita ogni volta.

Per chi sviluppa. Il nome del form non ammette spazi: è la chiave con cui viste e pagine lo cercano. Il disegno è in `frontend/ade-next/src/studio/form-builder`, il form a schermo in `frontend/ade-next/src/runtime/form`. Le funzioni che usano l'intelligenza artificiale e l'importazione di una maschera storica ci sono, ma la guida non le ha provate.

Per approfondire.

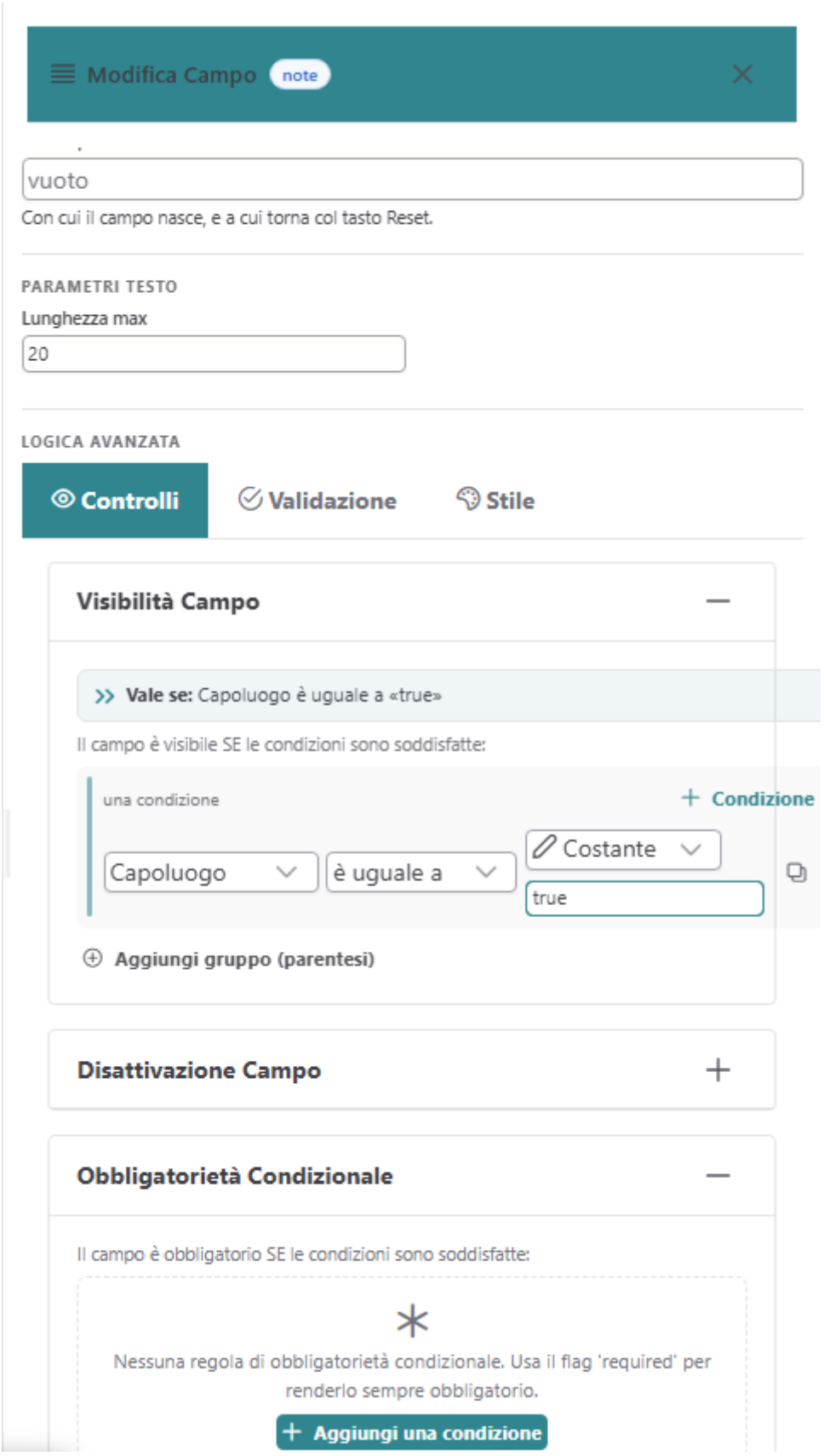
- Il Form Builder — disegnare un form e ritrovarlo uguale a runtime: `docs/guide/form/builder/README.md`

JE3.2 - Regole e controlli

In una frase. Le regole di un form si scelgono da un pannello, senza scrivere codice, e valgono due volte: mentre l'utente scrive e quando il server salva.

Un controllo che vive solo nella pagina si può aggirare, e uno che vive solo nel server si scopre troppo tardi. Qui la regola è una sola, scritta nel disegno del form. La leggono tutti e due, e danno lo stesso messaggio.

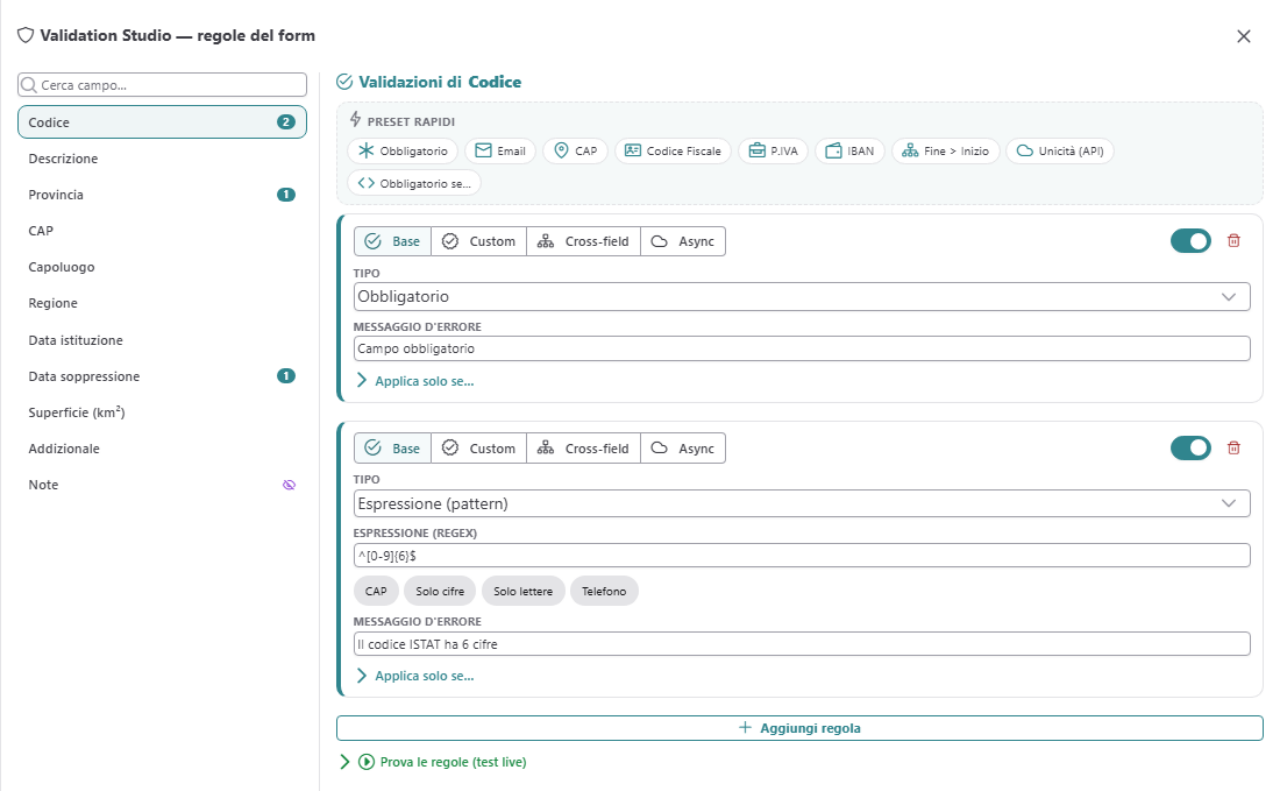
1. **Apri il campo.** Nel Form Builder, doppio clic sul campo. In fondo al pannello ci sono tre schede: **Controlli**, **Validazione**, **Stile**.
2. **Decidi quando il campo si vede.** Nella scheda **Controlli** componi la condizione scegliendo da tre tendine: un campo, un confronto, un valore. Sopra, la regola si rilegge in italiano. Allo stesso modo decidi quando il campo è bloccato e quando diventa obbligatorio.



La scheda Controlli: il campo Note si vede solo se Capoluogo è spuntato

3. **Aggiungi i controlli sul valore.** Nella scheda **Validazione** trovi quelli pronti: obbligatorio, email, CAP, codice fiscale, partita IVA, IBAN, «la fine dopo l'inizio», valore non già usato. Per ogni regola scrivi il messaggio che l'utente leggerà.
4. **Guardale tutte insieme.** La scheda **Validazioni** in fondo alla pagina apre l'elenco di tutti i campi con le loro regole. Ogni regola ha un interruttore: spenta, resta scritta ma non

vale.



Tutte le regole del form: a sinistra i campi, a destra le regole del campo scelto

5. **Prova.** Nella pagina, **Convalida** non è mai spento: se manca qualcosa dice cosa. Un campo nascosto non viene preteso.

> **Cosa ci guadagni.** Se un altro programma prova a salvare dati sbagliati saltando la pagina, il server li rifiuta con lo stesso messaggio che avrebbe visto l'utente.

> **Per chi sviluppa.** I due motori gemelli sono `frontend/ade-next/src/kernel/rule-engine` e, nel servizio, il pacchetto `validation`. Un file di casi in comune li tiene allineati. Un salvataggio rifiutato torna come errore 422.

> **Per approfondire.**
> • Il Form Builder — disegnare un form e ritrovarlo uguale a runtime: `docs/guide/form/builder/README.md`
> • Guida sviluppo — Validazioni, nascondimenti e picker delle relazioni: `docs/guide/sviluppo/validazioni-nascondimenti-e-picker.md`

JE3.3 - Scrivere valori e formule

In una frase. Ovunque serva un valore c'è la stessa casella con il pulsante **fx**: ti propone cosa puoi usare in quel punto e le funzioni pronte.

Quando disegni capita spesso di dover dire «prendi il valore da lì»: la provincia scritta nel form, il risultato del passo precedente, la data di oggi più trenta giorni. Scriverlo a memoria porta a errori che si vedono solo dopo. Il menu **fx** lo fa scegliere.

1. **Premi fx.** Lo trovi accanto alle caselle dei valori: nella condizione di un bivio, nei parametri di un endpoint, in «Esegui solo se».
2. **Scegli da dove viene il valore.** La prima parte del menu elenca quello che esiste in quel punto: i campi del form, i risultati dei passi precedenti, la riga corrente se sei dentro un ciclo. Quello che non esiste non compare.



Il menu fx: sopra i valori disponibili in quel punto, sotto le funzioni con un esempio

3. **Oppure cerca una funzione.** Le funzioni sono una trentina: testo, numeri, date, formato, logica. Cerchi quello che vuoi fare, non il nome: scrivendo «giorni» trovi la

funzione che aggiunge giorni a una data. Ognuna mostra un esempio con il risultato.

4. **Continua a scrivere.** La scelta entra dove avevi il cursore. Puoi comporre: una funzione attorno a un valore, due valori confrontati.
5. **Leggi l'errore, se c'è.** La formula viene controllata mentre scrivi. Se non si può leggere, sotto la casella compare cosa non va e in che punto.

> **Cosa ci guadagni.** Chi sa già cosa scrivere scrive, chi non lo sa sceglie. In tutti e due i casi il controllo è lo stesso che userà il motore quando esegue.

> **Per chi sviluppa.** I valori si scrivono fra doppie graffe, per esempio `{{in.Provincia}}`. Il motore delle formule è proprio del sistema (`frontend/ade-next/src/kernel/expression-engine`): non esegue codice libero. Gli esempi del menu sono eseguiti da un test.

> **Per approfondire.**
> • Scrivere valori — il controllo unico e le funzioni: `docs/guide/variabili/value-editor/README.md`

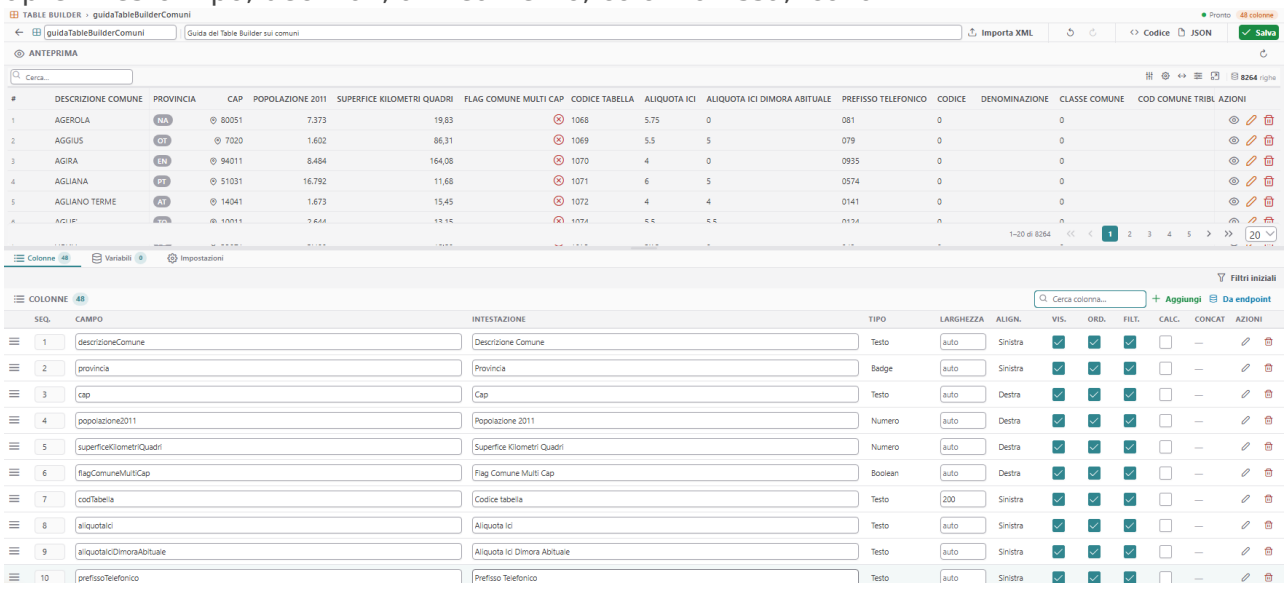
JE4 - Le tabelle

JE4.1 - Costruire una tabella

In una frase. Scegli da dove arrivano i dati, lasci che il Table Builder proponga le colonne e poi le sistemi una per una.

Una tabella è la cosa che si guarda di più in un gestionale. Disegnarla vuol dire decidere quali colonne mostrare e come: un numero con le migliaia, una data all'italiana, un sì o no come icona. Si decide una volta, sulla colonna.

1. **Apri il Table Builder.** Dal menu: **Low-Code Platform › Dynamic Tables › Table Builder**. Scrivi nome e descrizione.
2. **Scegli la sorgente.** Con **Sfoglia** cerchi l'endpoint che dà i dati. Poi **Genera colonne dall'endpoint**: il builder legge un campione e crea una colonna per ogni campo, indovinando il tipo.
3. **Sistema le colonne.** Nell'elenco in basso cambi intestazione e larghezza, e spunti se la colonna è visibile, ordinabile, filtrabile. Il numero nella prima casella la sposta. La matita apre il resto: tipo, decimali, allineamento, colonna fissa, icona.



Il Table Builder: sopra l'anteprima con i dati veri, sotto l'elenco delle colonne

4. **Aggiungi una colonna che nei dati non c'è.** Con **Aggiungi** crei una colonna calcolata da una formula, oppure una che unisce due campi con un separatore.
5. **Salva e guarda.** L'anteprima in alto è la tabella vera, con i dati veri.



#	DESCRIZIONE COMUNE	PROVINCIA	CAP	POPOLAZIONE 2011	SUPERFICE KILOMETRI QUADRI	FLAG COMUNE MULTI CAP	CODICE TABELLA	ALIQUOTA ICI	ALIQUO
1	AGEROLA	NA	80051	7.373	19,83		1068	5.75	0
2	AGGIUS	OT	7020	1.602	86,31		1069	5.5	5
3	AGIRA	EN	94011	8.484	164,08		1070	4	0
4	AGLIANA	PT	51031	16.792	11,68		1071	6	5
5	AGLIANO TERME	AT	14041	1.673	15,45		1072	4	4
6	AGLIE'	TO	10011	2.644	13,15		1074	5.5	5.5
7	AGNA	PD	35021	3.400	18,80		1075	5.75	0
8	AGNADELLO	CR	26020	3.757	12,08		1076	5	5
9	AGNANA CALABRA	RC	89040	569	8,49		1077	4.5	4.5
10	AGNONE	IS	86081	5.240	96,85		1080	5	0

La tabella nella pagina: la provincia come etichetta, i numeri con le migliaia, il sì o no come icona

Cosa ci guadagni. Nell'esempio la tabella dei comuni ha più di ottomila righe. La pagina ne chiede al server una manciata per volta, e ricerca e ordinamento li fa il server su tutte: la tabella resta veloce anche su archivi grandi.

Nota. Tutte le tabelle dell'applicazione, disegnate o no, usano la stessa griglia. Quello che impari su una vale per le altre.

Per chi sviluppa. Il disegno è in `frontend/ade-next/src/studio/table-builder`, la tabella a schermo in `frontend/ade-next/src/runtime/table`, la griglia comune in `frontend/core/src/grid` .

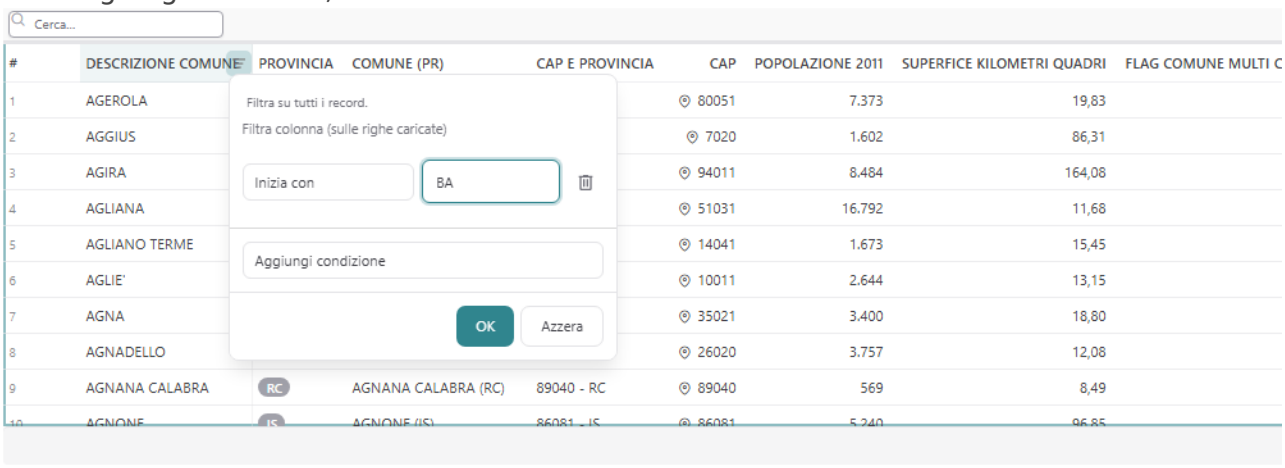
Per approfondire.
• Il Table Builder — ogni impostazione, e come si vede nella tabella: `docs/guide/tabella/builder/README.md`

JE4.2 - Filtri, gruppi e totali

In una frase. Decidi come si cerca nella tabella, per cosa si raggruppa e quali somme mostra in fondo.

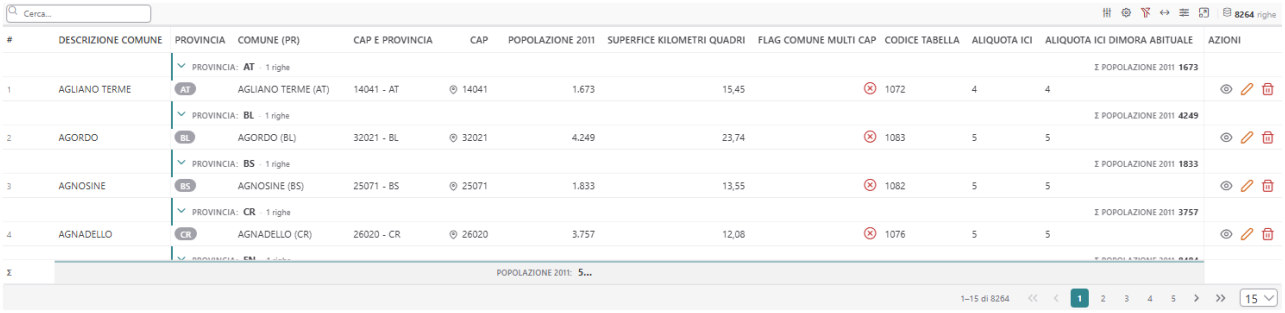
Una tabella lunga serve se ci trovi quello che cerchi e se ti dà il totale senza portare i dati in un foglio di calcolo. Queste tre cose si preparano nel disegno, e l'utente le può poi cambiare mentre lavora.

1. **Prepara i filtri.** Nel Table Builder, scheda **Impostazioni** › **Filtri**. La ricerca libera è già accesa. Per ogni colonna filtrabile scegli il modo di partenza, per esempio «Inizia con».
2. **Controlla chi filtra.** Con **Rileva cosa filtra l'endpoint** il builder chiede al servizio su quali colonne sa filtrare. Su quelle il filtro cerca in tutto l'archivio. Sulle altre cerca solo nelle righe già caricate, e la tabella lo dice.



Il filtro di una colonna: «Inizia con BA», e in alto la scritta che dice su cosa cerca

3. **Scegli il raggruppamento.** Nella scheda **Raggr.** indichi la colonna per cui la tabella si apre già raggruppata. Nell'esempio, la provincia.
4. **Aggiungi i totali.** Nella stessa scheda aggiungi le somme (o medie, conteggi, minimo, massimo) e accendi **Mostra Footer Totali**.



Righe raggruppate per provincia, con la somma della popolazione di ogni gruppo e il totale in fondo

5. **Decidi come si sfogliano le righe.** A pagine, a scorrimento continuo con **Carica altre**, oppure tutto in memoria per le tabelle piccole.

Cosa ci guadagni. Il totale in fondo è su tutte le righe trovate, non sulla pagina che hai davanti. E quando filtri, il contatore dice «33 di 569»: si vede che il filtro ha lavorato.

Nota. Chi usa la tabella trova gli stessi strumenti nella barra: filtri con «e» e «oppure», raggruppamento, colonne visibili, esportazione. Per numeri e grafici c'è il pulsante **Analisi**: vedi il libro JF - Analytics.

Per chi sviluppa. Verso il server i filtri viaggiano come `filter=campo:operatore:valore`, la ricerca libera come `q=`, l'ordinamento come `sort=`. Con i dati sul server l'ordinamento è su una colonna per volta.

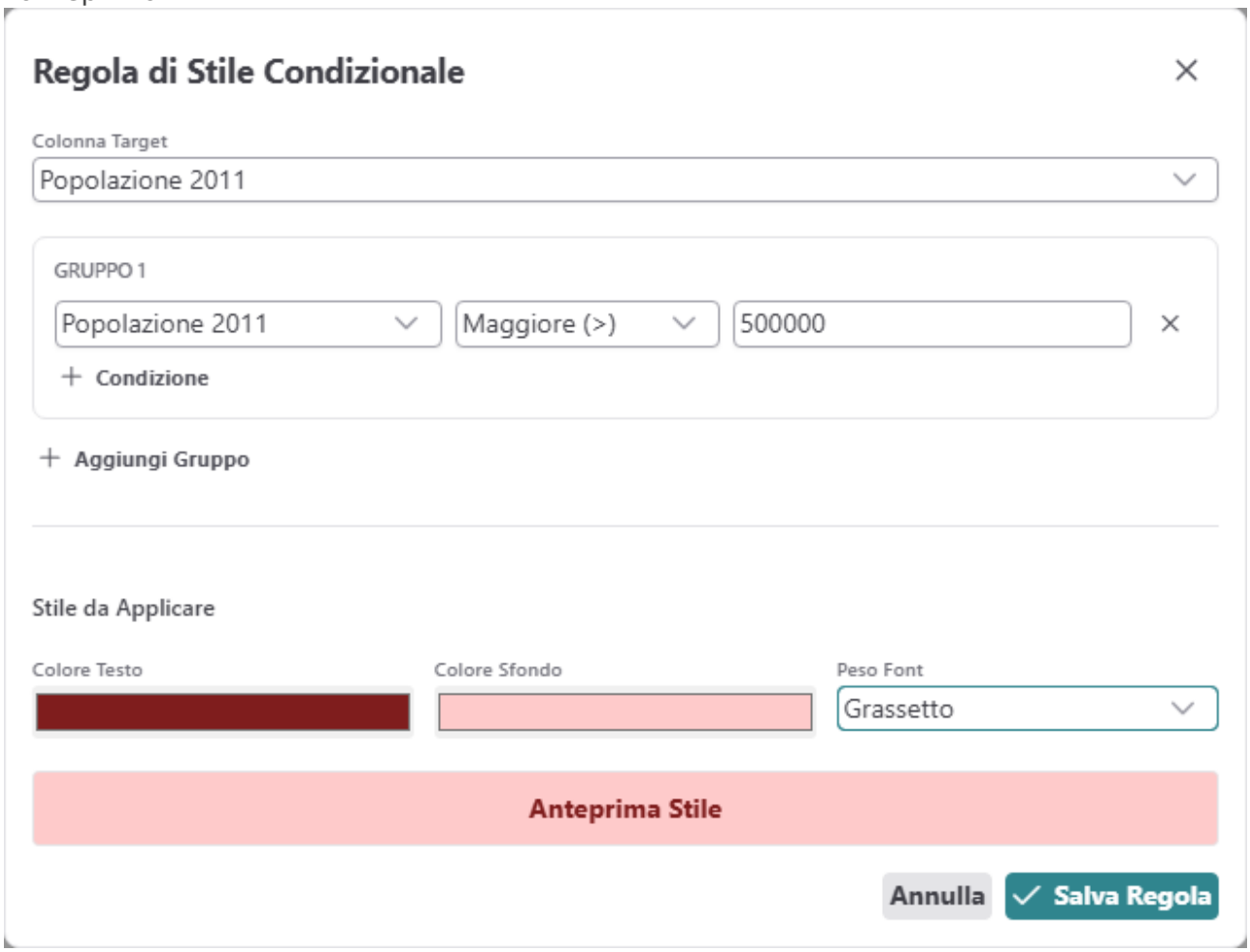
Per approfondire.
• Il Table Builder — ogni impostazione, e come si vede nella tabella: `docs/guide/tabella/builder/README.md`
• La tabella — una sola griglia, e cosa sa fare: `docs/guide/tabella/griglia/README.md`

JE4.3 - Stili e azioni

In una frase. Una regola colora le celle che contano, e i pulsanti di riga fanno partire quello che serve: aprire una scheda, modificare, cancellare.

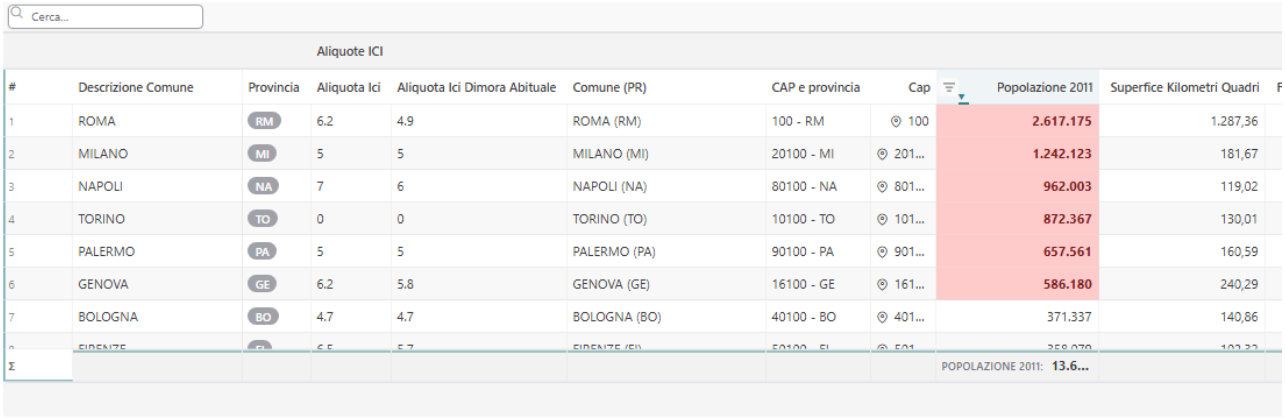
Una tabella non serve solo a leggere. Deve far vedere a colpo d'occhio cosa merita attenzione, e deve far arrivare con un clic alla schermata successiva.

1. **Scrivi una regola di stile.** Nel Table Builder, scheda **Impostazioni › Stile**. Scegli la colonna, la condizione e l'aspetto: colore del testo, sfondo, grassetto. In fondo vedi l'anteprima.



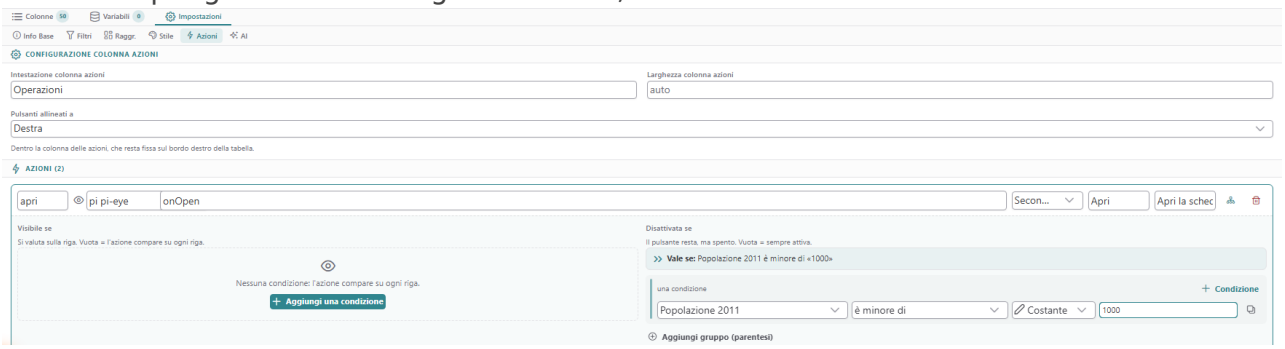
Una regola di stile: popolazione maggiore di 500.000, sfondo rosato e grassetto

2. **Guardala nella tabella.** La regola vale riga per riga, sui dati veri.



La regola nella tabella: le sei città sopra il mezzo milione sono evidenziate

3. **Aggiungi le azioni.** Nella scheda **Azioni** ogni azione è un pulsante sulla riga: un nome, un'icona, un colore. La tabella nata da un endpoint ne ha già tre: visualizza, modifica, elimina.
4. **Decidi quando l'azione si vede.** Ogni azione ha due regole: **Visibile se** e **Disattivata se**. Si compongono come le regole dei form, da tendine.



Le azioni: «apri» resta spenta sulle righe con popolazione sotto mille

5. **Collega l'azione a un'altra schermata.** Un'azione di riga può aprire un'altra funzione, in una finestra o a pagina intera, passandole i valori della riga. È il caso tipico della testata che apre le sue righe.

> **Cosa ci guadagni.** Colori e pulsanti seguono i dati senza una riga di programma: se domani la soglia cambia, cambi un numero nel disegno.

> **Nota.** Un'azione apre sempre una schermata, e le passa solo i valori che quella schermata dichiara di voler ricevere. Se ne manca uno il disegno lo segnala come errore, prima che se ne accorga l'utente.

> **Per chi sviluppa.** Nel flusso l'azione è un nodo «Apri schermata» collegato all'evento della riga. Dalla barra della tabella, il pulsante della stampa manda le righe a GPEnext: vedi il libro JK - GPEnext.

> **Per approfondire.**
> • Il Table Builder — ogni impostazione, e come si vede nella tabella:

`docs/guide/tabella/builder/README.md`

- Le azioni: cosa apre cosa: `docs/guide/flussi/azioni/README.md`

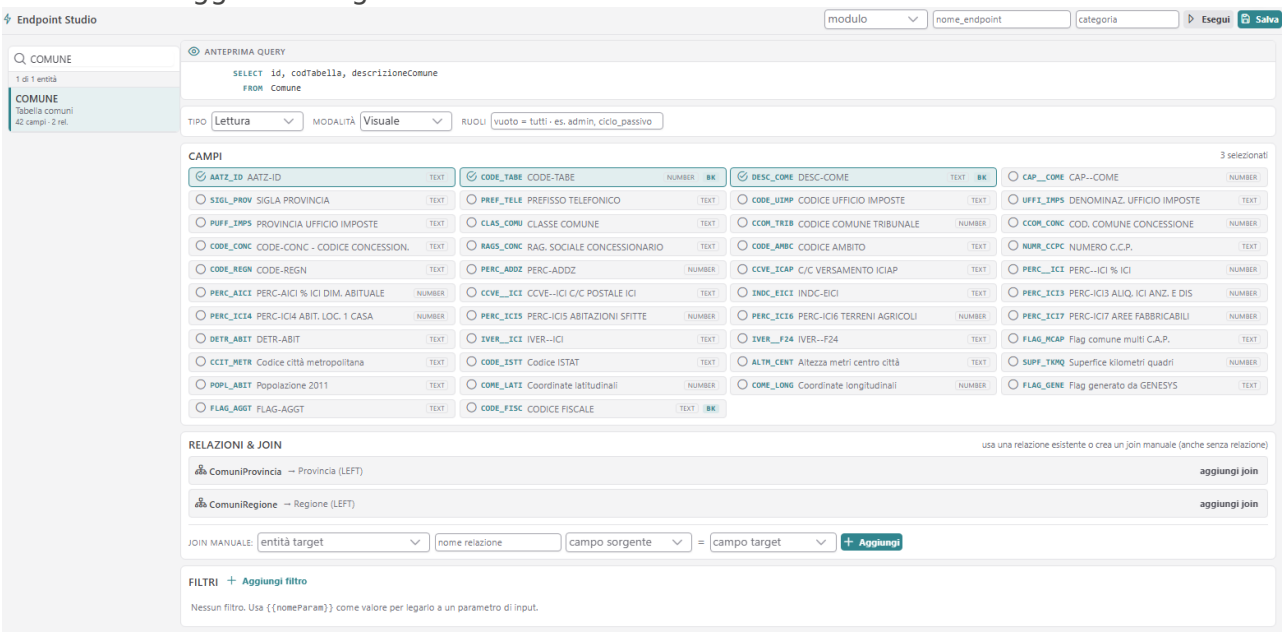
JE5 - Gli endpoint

JE5.1 - Leggere dati senza scrivere codice

In una frase. Un endpoint è una domanda ai dati con un nome: scegli l'archivio, i campi e i filtri, e la domanda è pronta per tabelle, form e flussi.

Per mostrare dei dati serve qualcuno che li vada a prendere. Di solito è un pezzo di programma scritto apposta. Nell'Endpoint Studio quella domanda la componi scegliendo dal catalogo, e la provi subito.

1. **Apri l'Endpoint Studio.** Dal menu: **Low-Code Platform › Dynamic Endpoints › Endpoint Studio**. A sinistra il catalogo, al centro il disegno, in alto il nome ed **Esegui**.
2. **Scegli l'archivio.** Cercalo nel catalogo a sinistra. Ogni riga dice quanti campi e quante relazioni ha.
3. **Scegli i campi.** Un clic su ogni campo che vuoi leggere. In cima, l'**anteprima** della domanda si aggiorna a ogni clic.



L'Endpoint Studio: tre campi scelti dall'archivio dei comuni, e sopra l'anteprima della domanda

4. **Aggiungi i collegamenti.** Sotto i campi trovi le relazioni dell'archivio, quelle del catalogo. Con **aggiungi join** leggi anche i campi dell'archivio collegato, per esempio la descrizione della provincia.
5. **Aggiungi filtri e parametri.** Un filtro può avere un valore fisso o un **parametro**: un valore che arriverà da fuori, per esempio dal form. Un parametro lasciato vuoto non filtra: la condizione viene tolta.

6. **Salva ed esegui.** Dai un nome, **Salva**, poi **Esegui**. Il risultato compare in fondo, in una tabella.

> **Cosa ci guadagni.** La stessa domanda la usano una tabella, una tendina di un form e un flusso. Se serve un campo in più lo aggiungi qui, una volta.

> **Nota.** Per i casi che il disegno non copre resta la modalità in cui si scrive l'SQL a mano, solo per leggere e con dei controlli. È la strada da usare per ultima.

> **Per chi sviluppa.** Ogni endpoint ha il suo indirizzo, mostrato in fondo allo studio nel riquadro **Invocazione**, e si può limitare ad alcuni ruoli. Gli strumenti sono in `frontend/ade-next/src/studio/endpoint-studio`. La domanda la esegue SIGEAnext.

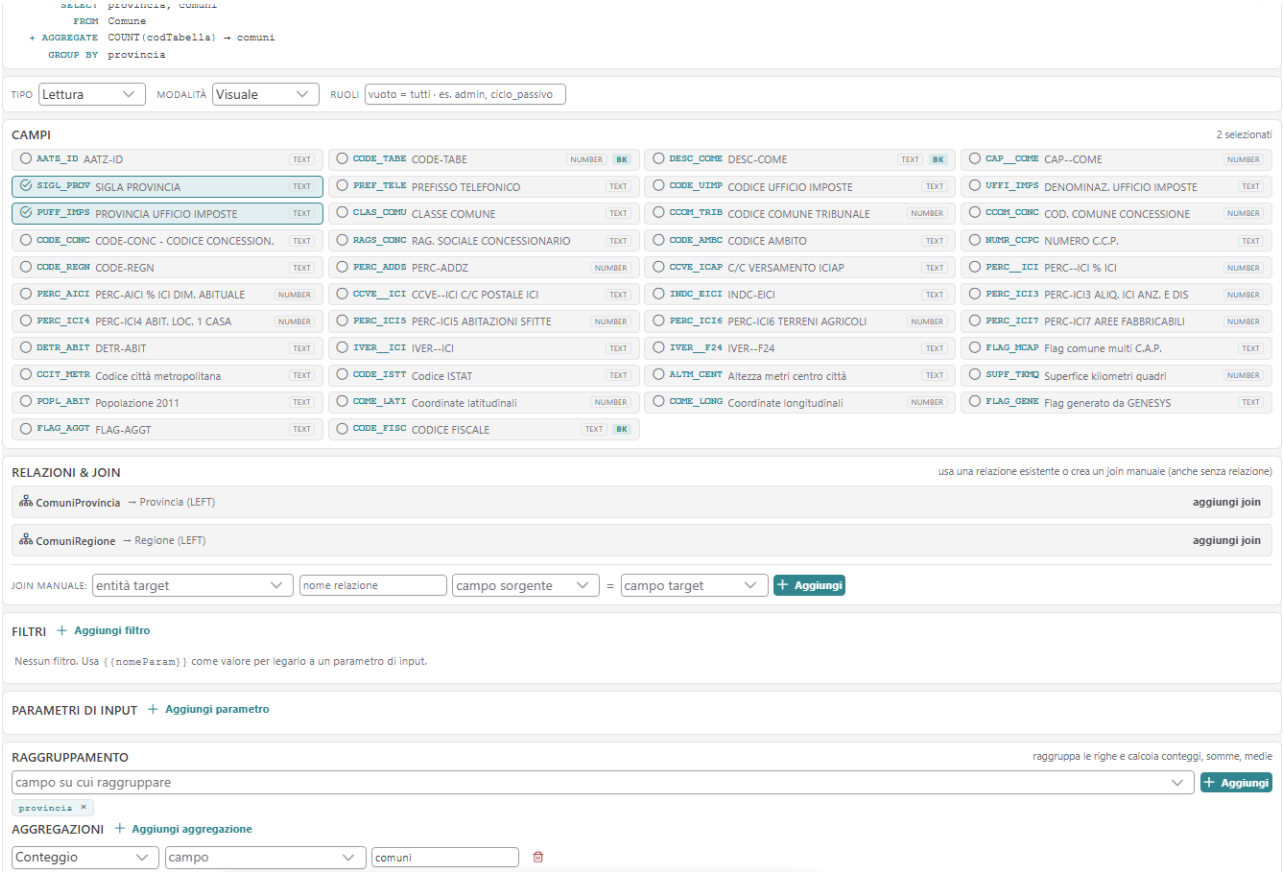
> **Per approfondire.**
> • Endpoint Studio — raggruppare e contare: `docs/guide/endpoint/endpoint-studio/README.md`

JE5.2 - Raggruppare e contare

In una frase. Lo stesso endpoint può rispondere con un riassunto invece che con le righe: quanti per gruppo, quale somma, quale massimo.

«Quanti comuni per provincia», «quale totale per mese», «l'importo più alto per deposito»: sono domande che danno una riga per gruppo. Prima finivano scritte a mano. Ora si disegnano negli stessi blocchi dell'Endpoint Studio.

1. **Scegli i campi.** Nell'Endpoint Studio, scegli l'archivio e i campi come per una lettura normale.
2. **Raggruppa.** Nel blocco **Raggruppamento** scegli il campo e premi **Aggiungi**. Compare come una pastiglia, che si toglie con la ×. Si può raggruppare solo per i campi che hai scelto di leggere.
3. **Aggiungi il conto.** Con **Aggiungi aggregazione** scegli la funzione (conteggio, conteggio dei diversi, somma, media, minimo, massimo), il campo e il nome della colonna che uscirà.



Comuni raggruppati per provincia, con il conteggio. In cima, l'anteprima dice che la domanda è un raggruppamento

4. **Tieni solo i gruppi che ti interessano.** Nel blocco **Solo i gruppi che…** metti una condizione sul risultato del conto: per esempio, solo le province con più di cinquanta

comuni. Un filtro normale qui non basta, perché lavora sulle righe prima che i gruppi esistano.

5. **Salva ed esegui.** Il risultato ha le colonne del gruppo più quelle calcolate.

> **Cosa ci guadagni.** Il conto lo fa il database, su tutto l'archivio, e alla pagina arriva solo il riassunto. Non si leggono migliaia di righe per poi sommarle a mano.

> **Nota.** L'anteprima in cima dice sempre se la domanda restituisce righe o gruppi. Se togli il raggruppamento, cambia subito.

> **Per chi sviluppa.** La condizione sui gruppi è la `HAVING` dell'SQL. Sui dati veri del server di prova: 63 province su 111 hanno più di 50 comuni.

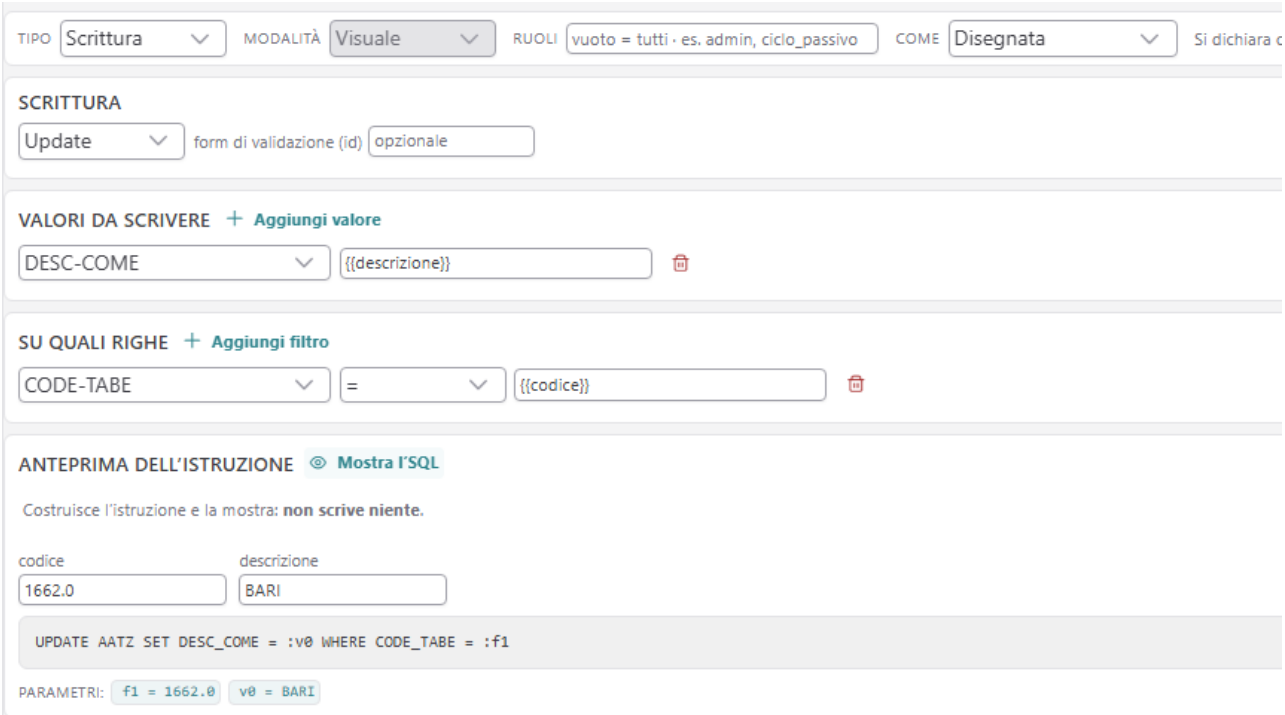
> **Per approfondire.**
> • Raggruppamenti, condizioni sui gruppi e ordinamento:
> `docs/guide/endpoint/aggregati/README.md`

JE5.3 - Scritture disegnate

In una frase. Anche inserire, modificare e cancellare si dichiara a blocchi: cosa scrivere, dove e su quali righe. L'istruzione la costruisce il motore, e la puoi guardare prima che parta.

Scrivere nei dati è la parte più delicata. Un'istruzione scritta a mano con un errore di battitura può toccare le righe sbagliate. Qui non scrivi l'istruzione: scegli archivio e campi dal catalogo, e il motore rifiuta le scritture pericolose.

1. **Scegli di scrivere.** Nell'Endpoint Studio, **Tipo › Scrittura**. Nella tendina **Come** resta **Disegnata**.
2. **Rispondi alle tre domande.**
   - **Scrittura**: inserire, modificare o cancellare, e su quale archivio.
   - **Valori da scrivere**: campo e valore. Il valore può essere un parametro che arriva da fuori.
   - **Su quali righe**: gli stessi filtri di una lettura.
3. **Guarda l'anteprima.** Premi **Mostra l'SQL**: il motore costruisce l'istruzione e la mostra, senza scrivere niente. Accanto metti dei valori di prova.



Una modifica disegnata: il valore, le righe da toccare e l'anteprima dell'istruzione, che non scrive niente

4. **Usala in un flusso.** Più scritture si mettono in fila in un flusso. Con **Scrive tutto o niente** passano tutte insieme, o non ne passa nessuna.

Il motore non fa partire:

- una modifica o una cancellazione **senza filtro**, perché toccherebbe tutte le righe;
- una scrittura su un campo che non esiste;
- una scrittura su un archivio che non è nel catalogo.

In tutti e tre i casi il messaggio dice a chi disegna cosa correggere.

> **Attenzione.** Le scritture disegnate sono spente finché chi gestisce l'installazione non le accende, e solo sugli archivi che elenca. Hanno i loro test, ma non sono ancora state usate su dati veri: si provano prima sull'ambiente di prova.

> **Cosa ci guadagni.** Di ogni scrittura resta scritto chi l'ha fatta. E le regole di cancellazione del catalogo valgono anche qui.

> **Per chi sviluppa.** Le variabili dell'installazione sono `LOWCODE_MUTATION_ENABLED` e `LOWCODE_MUTATION_TABLES`. Nell'istruzione i valori viaggiano sempre come parametri, mai come testo.

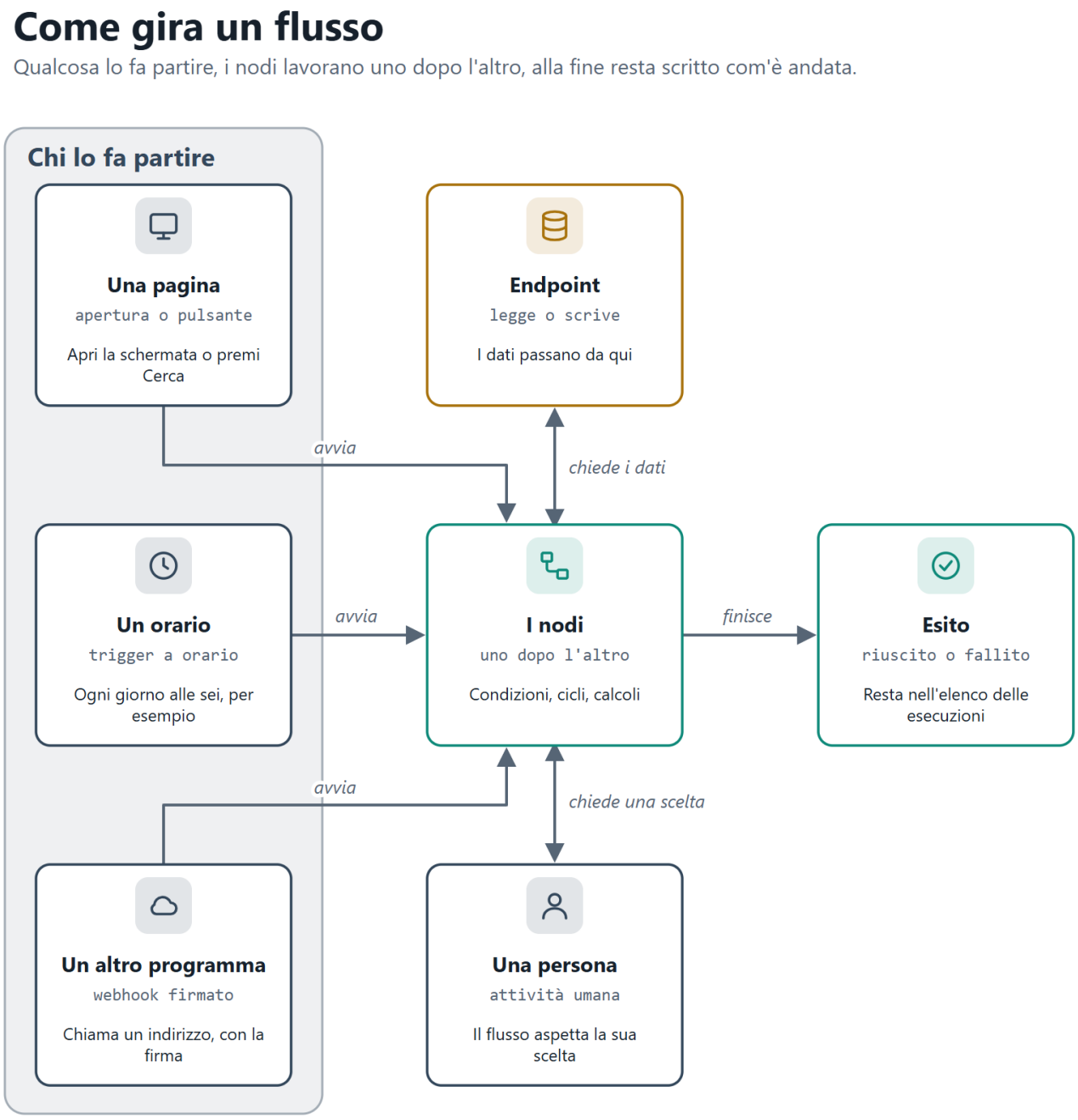
> **Per approfondire.**
> • Scritture disegnate: `docs/guide/endpoint/scritture-disegnate/README.md`

JE6 - I flussi

JE6.1 - Il primo flusso

In una frase. Un flusso dice cosa succede e in che ordine: lo disegni collegando dei nodi, e salvandolo nasce già la schermata.

Form, tabella ed endpoint da soli sono pezzi fermi. Il flusso li mette in moto: «quando si apre la pagina leggi i dati, quando l'utente preme Cerca rileggili con i suoi filtri, e metti le righe nella tabella».



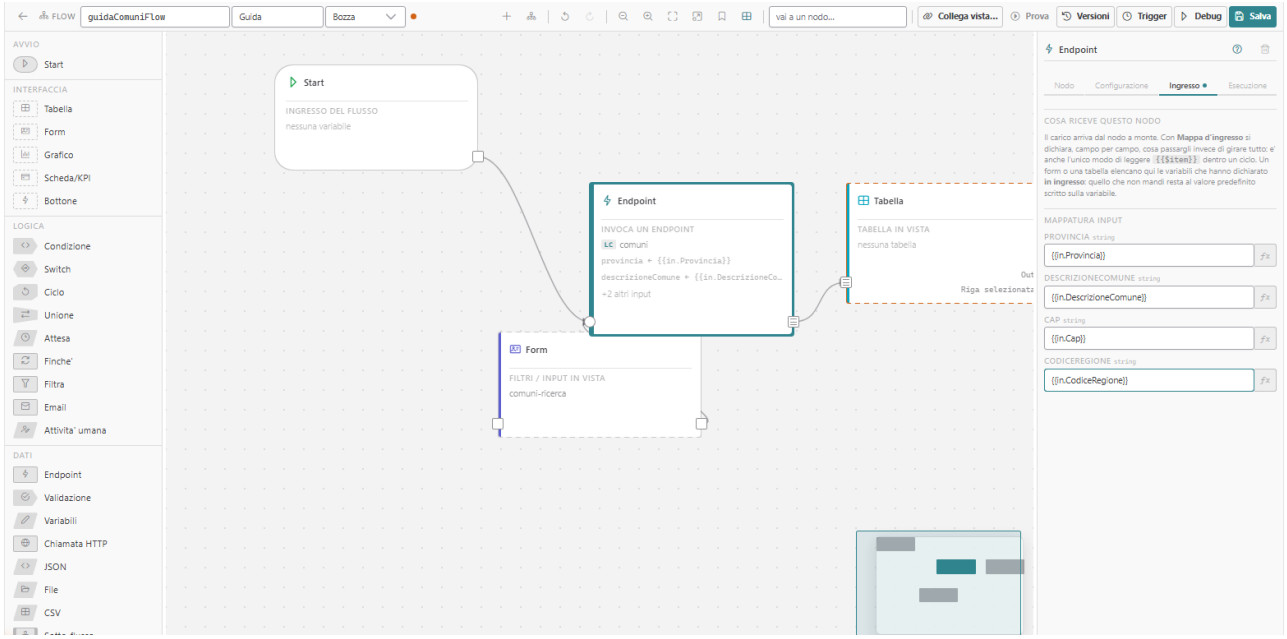
Come gira un flusso: qualcosa lo avvia, i nodi lavorano, alla fine resta l'esito

1. **Crea il flusso.** Dal menu: **Low-Code Platform** › **Dynamic Flows**, poi **Nuovo flusso**. Scegli una delle due strade.



Nuovo flusso: la strada guidata per le ricerche, il grafo libero per il resto

   - **Quick Flow Designer**: sei passi guidati per una ricerca completa, con filtri, risultati, grafico, azioni e voce di menu. Per consultare dati è la strada più corta.
   - **Flow Designer**: il disegno libero, nodo per nodo. I passi qui sotto seguono questa.
2. **Metti i nodi.** Trascinali dall'elenco a sinistra, o premi **Tab** e scrivi il nome. Per una ricerca ne bastano quattro: **Start**, **Form**, **Endpoint**, **Tabella**.
3. **Collegali.** Trascina dal quadratino a destra di un nodo a quello a sinistra del successivo. Se colleghi due nodi che non si capiscono, il bordo diventa rosso subito.
4. **Riempi ogni nodo.** Un clic sul nodo apre le proprietà a destra. Nel Form scegli quale form, nell'Endpoint quale endpoint e da dove prende ogni filtro, nella Tabella quale tabella.



Il Flow Designer: a sinistra i nodi, al centro il disegno, a destra le proprietà del nodo scelto

5. **Salva.** In basso la barra dice se il disegno regge. Salvando, la schermata nasce da sola: un riquadro per ogni nodo che si vede, il form sopra e la tabella sotto.
6. **Prova. Anteprima** apre la pagina vera, con i dati veri.

Cosa ci guadagni. A ogni salvataggio la schermata si riallinea al disegno. Aggiungi un nodo e compare il suo riquadro. Dove stanno i riquadri lo decidi tu, e resta.

Per chi sviluppa. Il nome del flusso è un identificatore: lettere, numeri, trattino basso. La schermata prende lo stesso nome con «View» in fondo. Il designer è in `frontend/ade-next/src/studio/flow-designer`, la strada guidata in `frontend/ade-next/src/studio/velocizzatore`.

Per approfondire.
• Flow Designer — guida a un flusso semplice: `docs/guide/flussi/flow-designer/README.md`
• Quick Flow Designer — una funzione con celle collegate, e funzioni che si riusano: `docs/guide/flussi/quick-flow-designer/README.md`

JE6.2 - Nodi e proprietà

In una frase. Ogni nodo fa una cosa sola, e le sue proprietà stanno in quattro schede: chi è, cosa fa, cosa riceve, come si comporta se va storto.

Un flusso si legge bene se ogni passo è piccolo e dice cosa fa. I nodi sono fatti per questo: ne scegli uno per ogni cosa da fare e lo regoli dal pannello a destra.

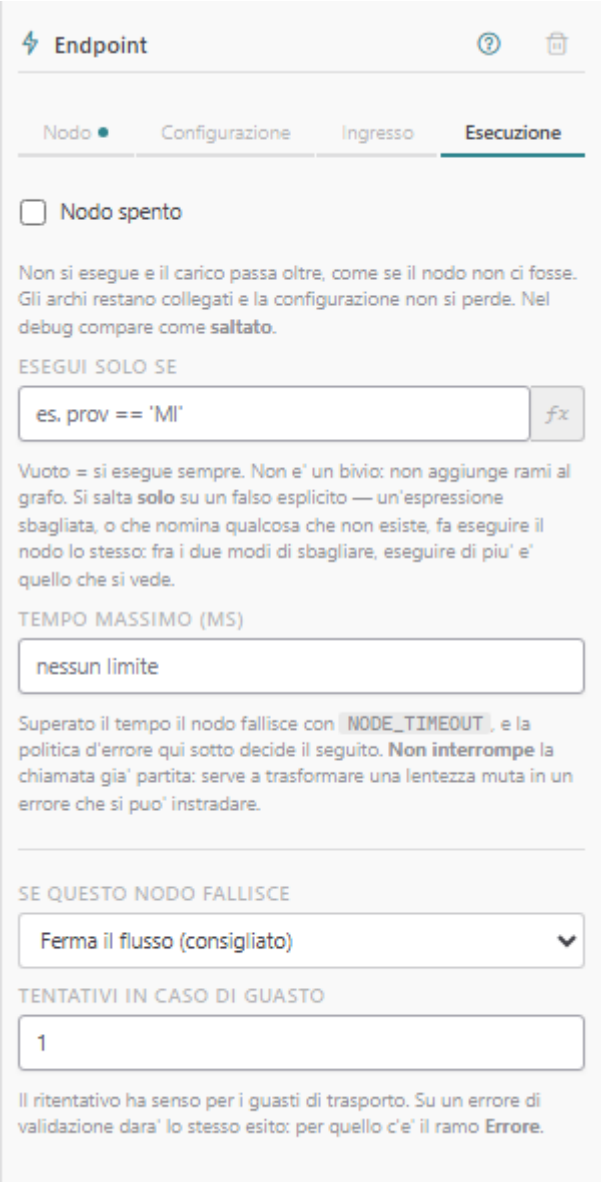
Le famiglie di nodi

- **Avvio**: Start, il punto da cui il flusso parte.
- **Interfaccia**: Tabella, Form, Grafico, Scheda, Bottone. Sono i pezzi che l'utente vede.
- **Logica**: Condizione, Switch, Ciclo, Finché, Unione, Attesa, Filtra, Email, Attività umana.
- **Dati**: Endpoint, Validazione, Variabili, Chiamata HTTP, JSON, File, CSV, Sotto-flusso.
- **Fine**: End, e Ferma con errore.

La forma del nodo dice la famiglia prima ancora dell'etichetta.

Le quattro schede

- **Nodo**: etichetta e descrizione. La descrizione non cambia niente nell'esecuzione: serve a chi leggerà il disegno fra sei mesi.
- **Configurazione**: cosa fa il nodo. È la scheda che si apre per prima.
- **Ingresso**: cosa riceve, campo per campo.
- **Esecuzione**: come si comporta.



La scheda Esecuzione: spegnere il nodo, eseguirlo solo a condizione, dargli un tempo massimo, decidere cosa fare se fallisce

Nella scheda **Esecuzione** ci sono le cose che servono quando si cerca un problema o si vuole un flusso robusto:

- **Nodo spento**: il passo viene saltato senza staccare i collegamenti. Nel debug resta visibile, segnato come saltato.
- **Esegui solo se**: una condizione. Se è falsa il nodo si salta.
- **Tempo massimo**: oltre quel tempo il nodo fallisce invece di restare appeso.
- **Se questo nodo fallisce** e **Tentativi**: fermare il flusso, riprovare, o proseguire su un ramo d'errore.

Un pallino sulla scheda dice che lì c'è qualcosa di diverso dal solito.

> **Nota.** La Chiamata HTTP verso servizi esterni ha limiti voluti, per sicurezza. L'Email parte solo se l'installazione ha configurato la posta. L'Attesa in un flusso normale arriva al

massimo a trenta secondi: per attese lunghe vedi Fermarsi e riprendere.

Per chi sviluppa. Il nome interno del nodo (per esempio `n_cerca`) è quello che ritrovi nel debug e nei messaggi d'errore. Cosa ogni nodo riceve e produce lo dichiara il servizio, ed è lo stesso giudizio usato al salvataggio.

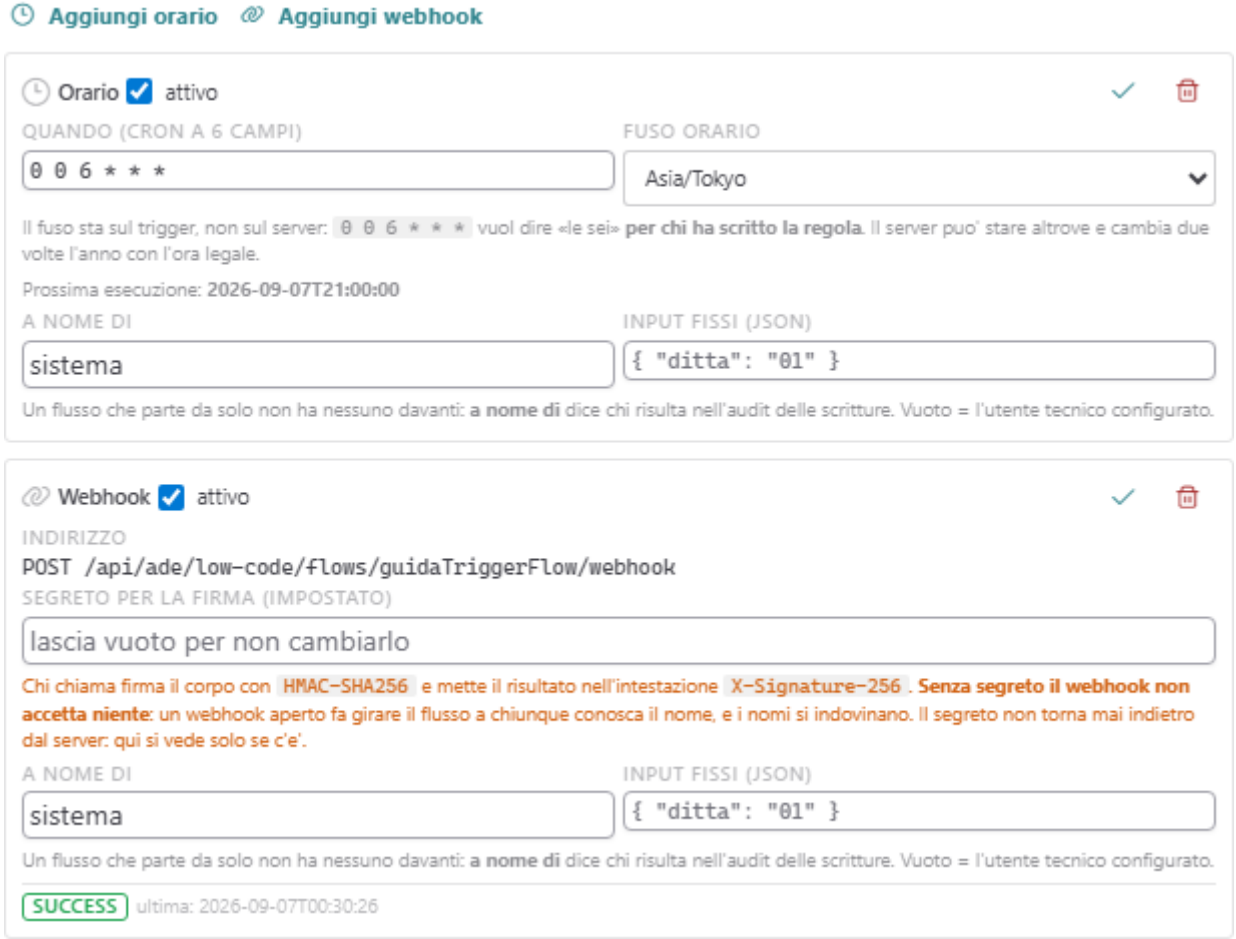
Per approfondire.
• Proprietà di nodo — l'inspector a schede: `docs/guide/flussi/flow-node-properties/README.md`
• Tre nodi nuovi — JSON, Attesa, Chiamata HTTP: `docs/guide/flussi/flow-nodes/README.md`

JE6.3 - Partire da solo: i trigger

In una frase. Un trigger fa partire un flusso senza che nessuno prema un pulsante: a un orario, oppure quando lo chiama un altro programma.

Certi lavori non hanno un utente davanti: un controllo ogni notte, un riepilogo ogni mattina, un ordine che arriva da un sito. Prima, per cambiare l'ora di un lavoro del genere serviva un programmatore. Ora è una riga nel pannello del flusso.

1. **Salva il flusso.** I trigger si attaccano a un flusso già salvato.
2. **Premi Trigger.** È nella barra del Flow Designer, accanto a **Debug**.
3. **Aggiungi un orario. Aggiungi orario** propone già «ogni giorno alle sei». L'orario si scrive in sei campi: secondi, minuti, ore, giorno, mese, giorno della settimana. Accanto scegli il **fuso orario**: le sei sono le sei di chi ha scritto la regola, non del server. Sotto leggi quando sarà la prossima esecuzione.
4. **Oppure aggiungi un webhook. Aggiungi webhook** ti dà un indirizzo. Il programma che lo chiama deve firmare la richiesta con un segreto che scegli tu.



Il pannello dei trigger: un orario con il suo fuso, un webhook con il segreto, e l'esito dell'ultima esecuzione

5. **Di' a nome di chi gira.** Il campo **A nome di** decide chi risulta come autore di quello che il flusso scrive. Vuoto, vale l'utente tecnico dell'installazione.

> **Attenzione.** Un webhook senza segreto non accetta niente. È voluto: l'indirizzo contiene il nome del flusso, e un nome si indovina. Il segreto, una volta scritto, non si può più rileggere dalla pagina.

> **Nota.** Con il login acceso, i flussi a orario chiamano gli altri servizi con un'identità vera. Questa parte è scritta ma non ancora provata dal vivo su un'installazione.

> **Cosa ci guadagni.** L'ora di un lavoro automatico la cambia chi disegna il flusso, in un minuto. E ogni partenza lascia un esito che si può andare a guardare.

> **Per chi sviluppa.** L'orario è un cron a sei campi. La firma del webhook è `HMAC-SHA256` del corpo della richiesta, nell'intestazione `X-Signature-256`. Firma assente, sbagliata o segreto mancante danno lo stesso messaggio.

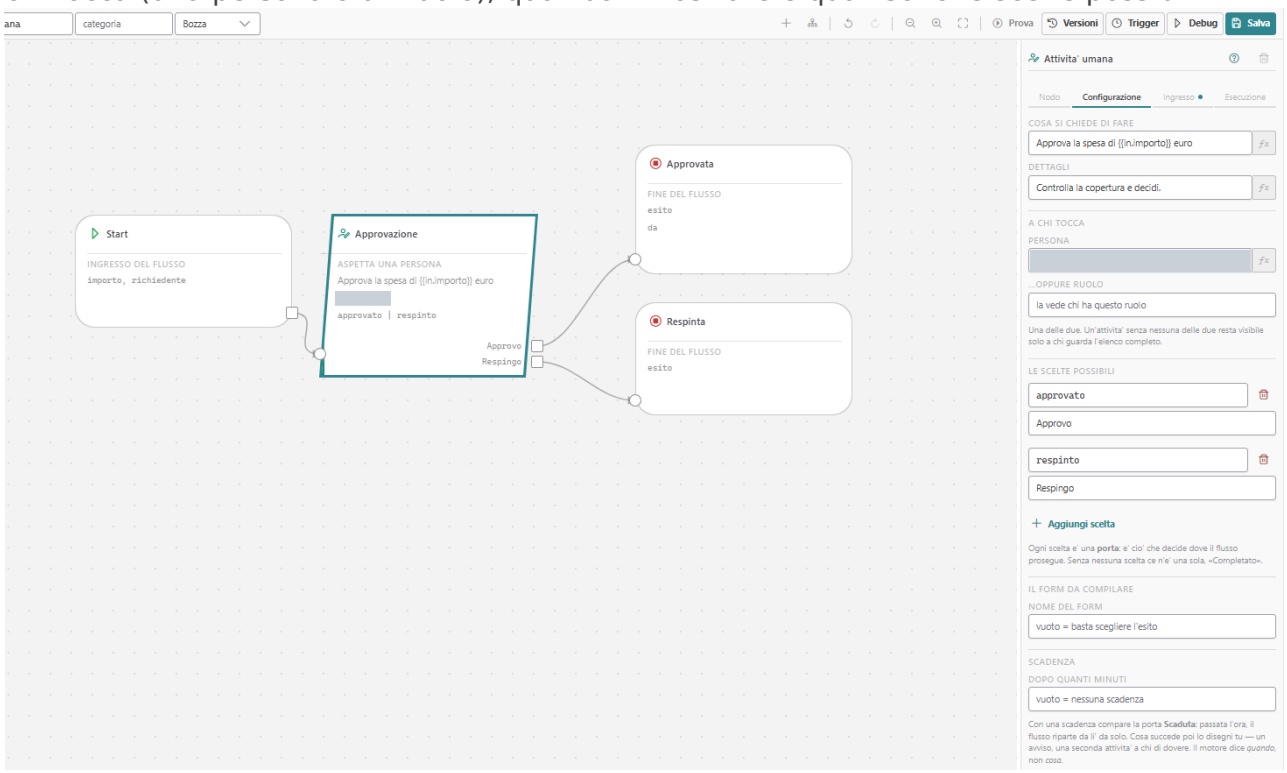
> **Per approfondire.**
> • Trigger — come un flusso parte da solo: `docs/guide/flussi/flow-triggers/README.md`

JE6.4 - Fermarsi e riprendere

In una frase. Un flusso può fermarsi ad aspettare una persona o una scadenza, anche per giorni, e ripartire da dove era rimasto.

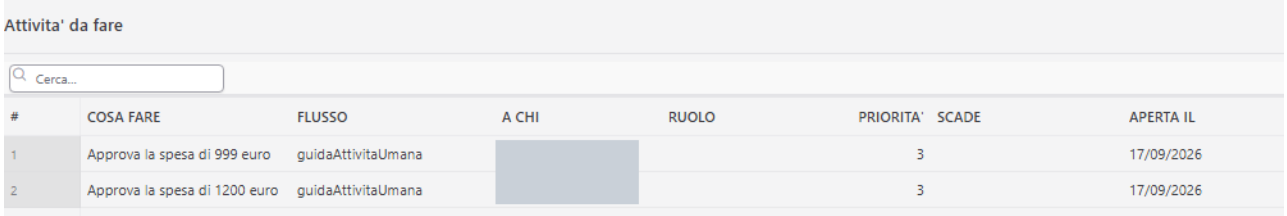
Un'approvazione non si chiude in un secondo. Qualcuno deve guardare la richiesta e decidere, magari domani. Il nodo **Attività umana** serve a questo: ferma il flusso, mette la richiesta nell'elenco di chi deve decidere, e riparte quando arriva la risposta.

1. **Metti il nodo Attività umana.** È fra i nodi di Logica. Nel pannello scrivi cosa si chiede, a chi tocca (una persona o un ruolo), quali dati mostrare e quali sono le scelte possibili.



Un'approvazione: il flusso si ferma sul nodo e prosegue su «Approvo» o su «Respingo»

2. **Collega ogni scelta.** Ogni scelta diventa un'uscita del nodo. «Approvo» porta da una parte, «Respingo» dall'altra.
3. **Avvialo in secondo piano.** Nella finestra del **Debug**, **Esegui in background**. Non aspetti la fine: ricevi il numero dell'esecuzione, con cui la segui.
4. **Chi deve decidere apre Attività da fare.** Trova le richieste che toccano a lui o al suo ruolo. Aprendone una vede i dati su cui decide e un pulsante per ogni scelta.



Attività da fare: le richieste in attesa, con cosa si chiede e in quale flusso

5. **Il flusso riparte.** Dopo la scelta prosegue sul ramo giusto. Una seconda risposta alla stessa attività viene rifiutata.

Nell'elenco **Esecuzioni dei flussi** le esecuzioni ferme si vedono: stato **In attesa**, con la colonna **Fermo su** (quale nodo) e **Riprende** (quando).

> **Cosa ci guadagni.** Approvazioni, scadenze e lavori lunghi si disegnano come ogni altro passo. Il flusso fermo resta scritto nell'archivio e riprende da dove era, non da capo. Finché aspetta si può anche annullare.

> **Nota.** L'attività può avere una scadenza in minuti. Passata l'ora, il flusso riparte da un'uscita apposta, **Scaduta**: cosa succede dopo lo disegni tu.

> **Per chi sviluppa.** L'avvio in secondo piano risponde subito con il numero dell'esecuzione e l'indirizzo da cui leggerne lo stato. Chi vede un'attività si decide per nome utente e per ruolo.

> **Per approfondire.**
> • Esecuzione durevole — un flusso che si ferma e riprende: `docs/guide/flussi/flow-durevole/README.md`

JE6.5 - Versioni e ripristino

In una frase. Ogni salvataggio che cambia il disegno lascia una versione: puoi confrontarne due e tornare a quella che funzionava.

Un disegno si rompe in modi che non si notano subito: un collegamento staccato per sbaglio, una condizione girata al contrario. Te ne accorgi il giorno dopo, quando non ricordi più com'era. Con le versioni non serve ricordare.

1. **Premi Versioni.** È nella barra del Flow Designer, accanto a **Trigger**. Si apre la cronologia, dalla più recente.
2. **Leggi la cronologia.** C'è una riga per ogni salvataggio che ha cambiato davvero qualcosa. Spostare un nodo di poco o ritoccare la descrizione non lascia una versione: così l'elenco resta corto e si trova quella giusta.
3. **Confronta.** Scegli due versioni e premi **Confronta**. La risposta parla del disegno: quali nodi e quali collegamenti sono stati aggiunti, tolti o modificati.



Il confronto fra due versioni: un nodo aggiunto, uno modificato, un collegamento aggiunto

4. **Ripristina.** Premi **Ripristina** sulla versione a cui vuoi tornare. Prima di farlo il sistema salva una versione di com'era adesso: se hai scelto quella sbagliata, torni indietro anche da lì.
5. **Esporta, se serve. Esporta** scarica il disegno in un file, per conservarlo o portarlo su un'altra installazione. Password e segreti non finiscono nel file.

Cosa ci guadagni. Si può provare una modifica senza paura. Anche form, tabelle ed endpoint tengono le loro versioni, allo stesso modo: la versione di prima viene messa da parte a ogni salvataggio che cambia qualcosa.

Nota. Il ripristino non cambia il **nome** del flusso. Il nome è quello con cui lo chiamano le schermate e gli altri flussi: riportarlo indietro li lascerebbe senza.

Per chi sviluppa. Il confronto è fatto sulla struttura del disegno, non sul testo, e ignora date e autore. Per form, tabelle ed endpoint le versioni si leggono dal servizio; il pannello con confronto e ripristino descritto qui è quello del Flow Designer.

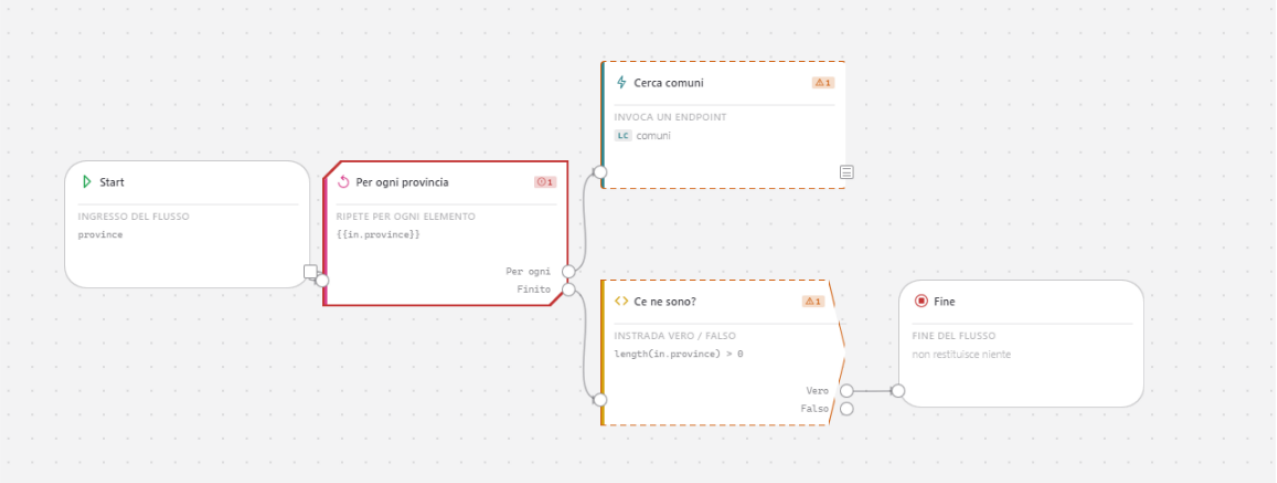
Per approfondire.
• Versioni del flusso — cronologia, confronto, ripristino, export: `docs/guide/flussi/flow-versions/README.md`

JE6.6 - Trovare gli errori

In una frase. Il designer ti dice mentre disegni cosa non va e dove, e il Debug ti mostra passo per passo cosa è successo.

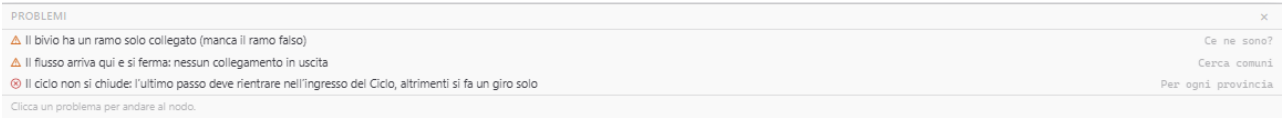
Un disegno sbagliato è peggio di un programma sbagliato se non te ne accorgi. Per questo il controllo è continuo: non devi salvare né eseguire per sapere che manca qualcosa.

1. **Guarda i nodi.** Un nodo con un problema ha il bordo colorato e un numero. Tratteggiato arancione: c'è qualcosa da completare. Rosso: c'è un errore.



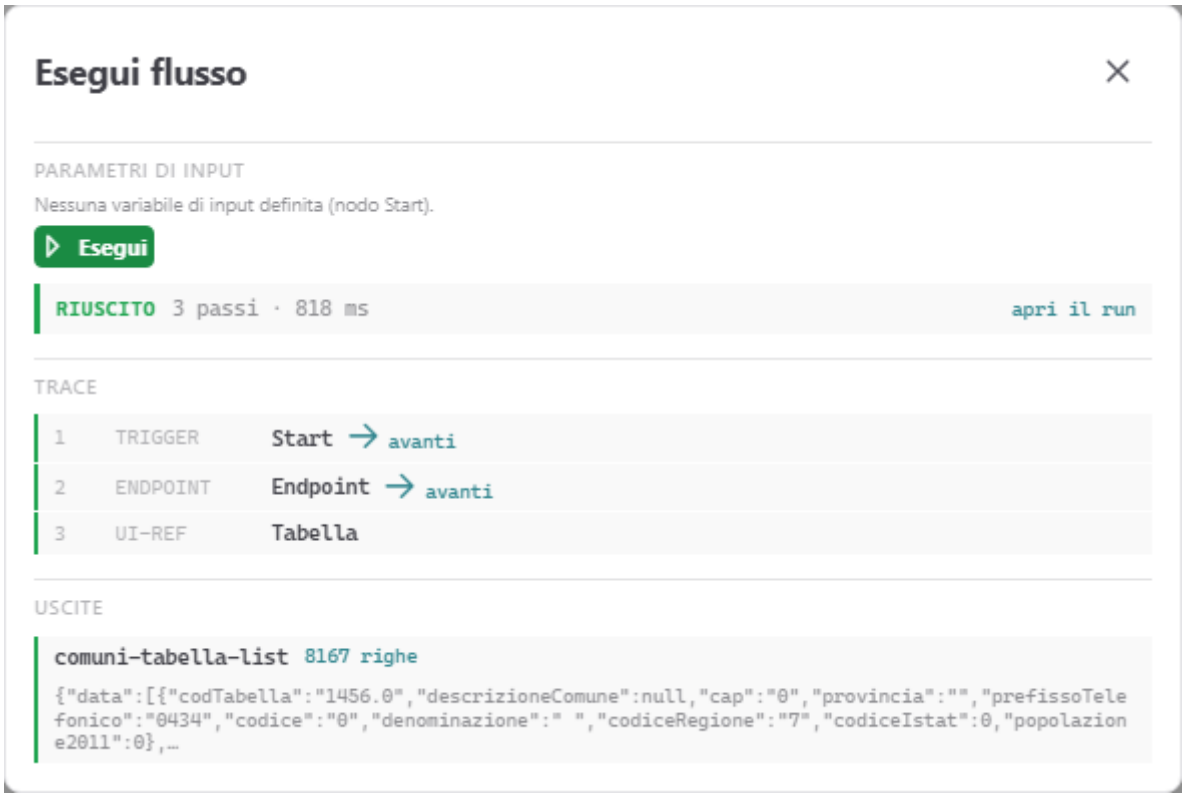
Un flusso con tre problemi: il ciclo ha un errore, la ricerca e il bivio un avviso

2. **Apri l'elenco.** In basso a destra la barra conta errori e avvisi e, se c'è un errore, aggiunge «non pubblicabile». È un pulsante: premilo e leggi i problemi, in italiano, ciascuno con il nome del nodo.



L'elenco dei problemi: cosa non va, e su quale nodo

3. **Clicca un problema.** Il disegno si sposta sul nodo e apre la scheda dove si corregge.
4. **Correggi.** Con un **errore** il flusso non si può pubblicare. Un **avviso** non blocca: segnala qualcosa che potrebbe essere voluto, come un bivio con un ramo solo.
5. **Esegui il Debug. Debug** fa girare il flusso e mostra l'elenco dei passi: quali nodi sono partiti, in che ordine, cosa hanno prodotto, quanto tempo è servito. Se un passo fallisce, dice quale e perché.



Il Debug: riuscito in tre passi, con il percorso seguito e i dati usciti

I controlli sono una ventina. Tre riguardano la forma del disegno, e sono quelli che ingannano di più: un ciclo che non torna indietro (farebbe un giro solo), un bivio con un ramo solo, un passo da cui non esce niente.

> **Cosa ci guadagni.** Gli errori si trovano prima di consegnare la funzione, non quando la usa il cliente. E lo stesso controllo lo rifà il servizio al salvataggio: non si può aggirare.

> **Nota.** Se l'anteprima o il Debug mostrano un errore sui dati, non sono dati finti: vuol dire che in quel momento SIGEAnext non raggiunge i Dati aziendali.

> **Per chi sviluppa.** Il controllo del disegno gira uguale nell'app web e nel servizio, tenuto allineato da un file di casi in comune. Ogni problema ha un codice, per esempio `LOOP_NOT_CLOSED`.

> **Per approfondire.**
> • I problemi del grafo — vederli, capirli, raggiungerli: `docs/guide/flussi/graph-problems/README.md`

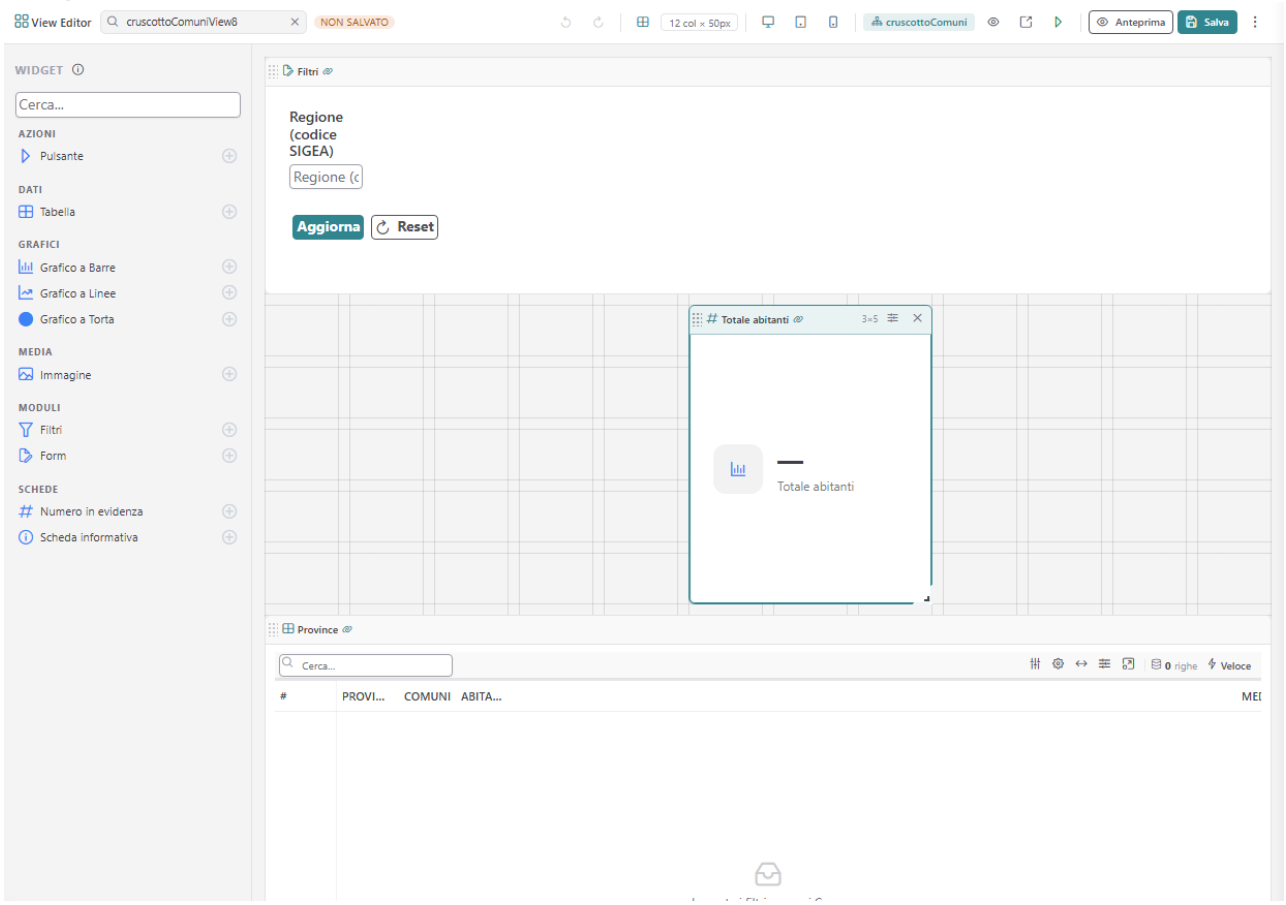
JE7 - Viste e variabili

JE7.1 - Mettere insieme form, tabelle e grafici

In una frase. La vista è la schermata che l'utente apre: nel View Editor disponi i pezzi su una griglia e li leghi al flusso che li fa lavorare.

Form, tabelle e grafici si disegnano uno per uno. La vista è il foglio su cui si mettono insieme: i filtri in alto, un numero in evidenza, la tabella sotto, un grafico di fianco.

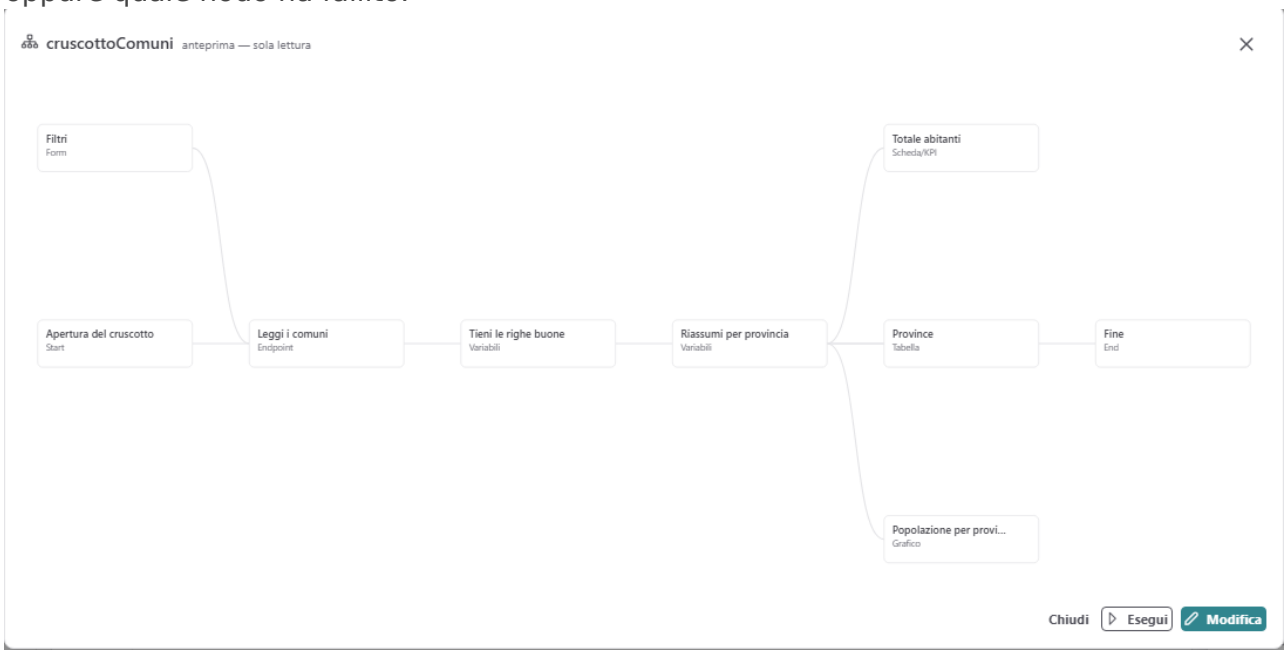
1. **Apri il View Editor.** Dal menu: **Low-Code Platform › Dynamic Views › View Builder**. Se la vista è nata salvando un flusso, la trovi già con i suoi pezzi: ti resta da sistemarli.
2. **Aggiungi i pezzi.** A sinistra c'è l'elenco: pulsante, tabella, grafico a barre, a linee o a torta, immagine, filtri, form, numero in evidenza, scheda informativa. Qui si chiamano **widget**.



Il View Editor: a sinistra i pezzi, al centro la schermata con filtri, un numero in evidenza e una tabella

3. **Disponili.** La griglia ha dodici colonne. Trascini un pezzo per spostarlo, tiri l'angolo in basso a destra per ridimensionarlo. Un segnaposto ti mostra dove si poserà. Un pulsante accende le linee della griglia quando devi allineare.

4. **Prova le larghezze.** Nella barra scegli la larghezza di prova: schermo grande, tablet, telefono.
5. **Guarda il flusso collegato.** Se la vista ha un flusso, la barra ne mostra il nome. Da lì lo apri nel designer o lo esegui. L'esito arriva come messaggio: quanti passi, quanto tempo, oppure quale nodo ha fallito.



Il flusso dietro una schermata: filtri, lettura dei dati, e poi numero, tabella e grafico

6. **Anteprima e Salva. Anteprima** mostra la schermata con i dati veri.

> **Cosa ci guadagni.** Il flusso decide quali pezzi esistono, tu decidi dove stanno. Cambiare l'aspetto di una schermata non tocca la sua logica, e viceversa.

> **Nota.** Una vista può dichiarare dei **parametri di schermata**: i valori che vuole ricevere quando un'altra schermata la apre, per esempio il numero dell'ordine. Per analizzare i dati con numeri e grafici già pronti c'è Analytics: vedi il libro JF - Analytics.

> **Per chi sviluppa.** Una vista ha al più un flusso, e un flusso al più una vista. I grafici dentro una vista sono di cinque tipi. Un contenitore a schede dentro la vista non c'è. Il codice è in `frontend/ade-next/src/studio/view-editor` e `frontend/ade-next/src/runtime/view`.

> **Per approfondire.**
> • View Editor — comporre una schermata: `docs/guide/viste/view-editor/README.md`

JE7.2 - Le variabili dei componenti

In una frase. Una variabile è un valore con un nome che un form o una tabella conosce: può leggerlo da un endpoint, riceverlo da fuori, averlo fisso o calcolarlo.

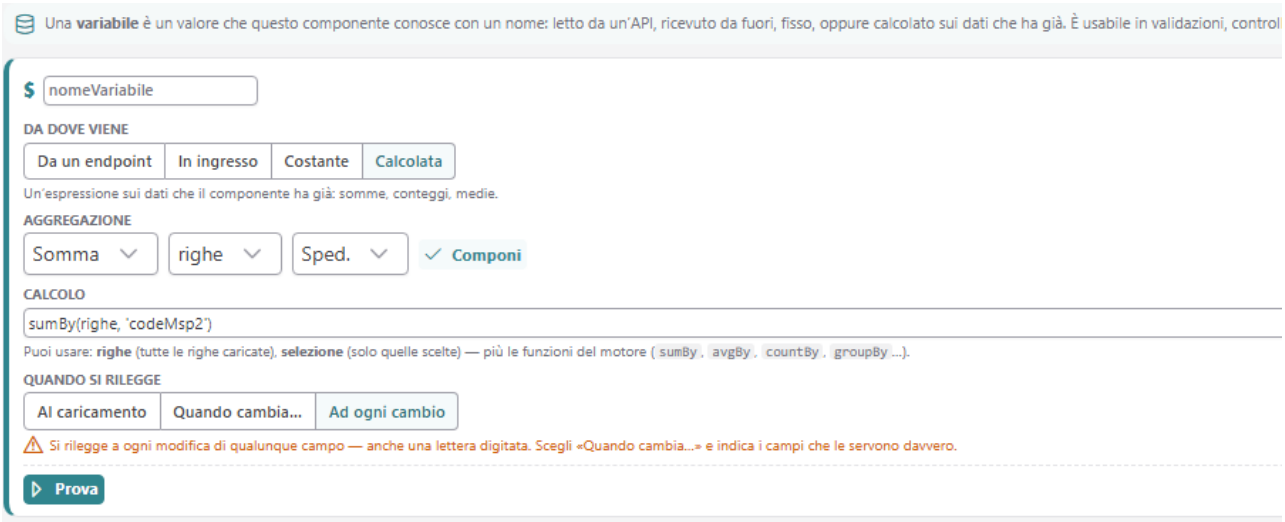
Le variabili servono a far parlare i pezzi fra loro. Una tendina che si riempie con le regioni lette dai dati, un form che riceve il numero dell'ordine dal flusso, il totale di una colonna: sono tutte variabili.

1. **Apri la scheda Variabili.** C'è nel Form Builder e nel Table Builder. **Aggiungi variabile** e dalle un nome.
2. **Scegli da dove viene.**
   - **Da un endpoint**: legge il valore dai dati.
   - **In ingresso**: arriva da fuori, dal flusso o dalla schermata che apre questa. Puoi dire quanto vale se non arriva, e se è obbligatoria.
   - **Costante**: un valore fisso, deciso qui.
   - **Calcolata**: una formula su quello che il componente ha già.



Una variabile letta da un endpoint: da dove viene, quando si rilegge, e la struttura trovata con Prova

3. **Premi Prova.** Per una variabile letta da un endpoint, **Prova** mostra cosa arriva: ogni campo con il tipo e un esempio del valore vero. Se la lettura fallisce lo dice, con il motivo.
4. **Scegli quando si rilegge. Al caricamento** (una volta), **Quando cambia…** (solo quando cambiano i campi che indichi), **Ad ogni cambio**. Il secondo è quasi sempre quello giusto: il terzo rilegge a ogni lettera digitata.
5. **Usala.** Nel form una variabile riempie le opzioni di una tendina o entra in una regola. Nella tabella una variabile calcolata vede le righe caricate e quelle scelte: così fai la somma di una colonna.



Una variabile calcolata in una tabella: la somma di una colonna sulle righe caricate

Cosa ci guadagni. I valori non si scrivono a mano: si scelgono da un elenco che conosce i nomi veri. Un nome sbagliato che «sembra giusto» era la causa più frequente di schermate vuote.

Nota. Con la casella **in uscita** una variabile si può leggere anche dal flusso che pilota il componente.

Per chi sviluppa. In una tabella la formula vede `righe` e `selezione`, e funzioni come `sumBy(righe, 'importo')`. Nei flussi i valori dell'installazione si leggono come `{{$site.chiave}}`, quelli del ciclo come `{{$item}}`.

Per approfondire.
- Le variabili dei componenti disegnati: `docs/guide/variabili/variabili-componenti/README.md`
- Variabili e condizioni — lo standard: `docs/guide/variabili/variabili-e-condizioni/README.md`

JE7.3 - Se qualcosa non va

In una frase. I problemi più comuni quando si disegna con ADEnext, come «vedo questo, faccio questo», e le parole da sapere.

Quasi sempre lo strumento dice già cosa non va: un bordo rosso, un avviso sotto la casella, un messaggio con il motivo. Qui trovi cosa fare dopo averlo letto.

Dati e form

- **Non vedo le voci di menu per disegnare.** Serve il livello Utente avanzato: chiedilo a chi gestisce gli accessi.
- **La tendina di un form è vuota.** Se le opzioni arrivano da una variabile, apri la scheda **Variabili** e premi **Prova**: dice se la lettura riesce e cosa arriva.
- **Una regola non scatta.** Apri la scheda **Validazioni** del form e guarda l'interruttore della regola: spenta, resta scritta ma non vale.
- **Il codice di un elenco chiuso non si cambia.** È voluto: è già scritto nei dati. Cambia l'etichetta, oppure disattiva il valore e creane uno nuovo.

Tabelle

- **Il filtro trova meno righe di quelle che mi aspetto.** Su quella colonna il filtro cerca solo nelle righe già caricate. Nel Table Builder premi **Rileva cosa filtra l'endpoint**.
- **La larghezza che ho scritto non viene rispettata.** In modalità «Adatta» la larghezza è una preferenza. Passa a **Espandi**: vale alla lettera e la tabella scorre.
- **La tabella si apre vuota.** Controlla se **Carica all'apertura** è spento: in quel caso aspetta la prima ricerca.

Endpoint

- **Esegui o Anteprima mostrano un errore sui dati.** SIGEAnext in quel momento non raggiunge i Dati aziendali. Avvisa chi gestisce il server.
- **Una scrittura viene rifiutata.** Leggi il motivo: le scritture sono spente sull'installazione, manca il filtro, oppure l'archivio non è nel catalogo. Vedi Scritture disegnate.

Flussi e viste

- **La barra dice «non pubblicabile».** Premi sul conteggio in basso a destra e correggi gli errori: vedi Trovare gli errori.
- **Il pulsante Trigger o Versioni non apre niente.** Il flusso va prima salvato.
- **Il webhook risponde sempre di no.** Manca il segreto, o la firma non torna. Il messaggio è lo stesso nei due casi, apposta.
- **Un flusso sembra fermo.** Aprilo nell'elenco **Esecuzioni dei flussi**: se è **In attesa**, la colonna **Fermo su** dice dove, e dal dettaglio si può annullare. Di solito aspetta una persona, in **Attività da fare**.
- **Ho salvato e non c'è una versione nuova.** Il salvataggio non ha cambiato il disegno.
- **Ho rotto un flusso che funzionava. Versioni**, poi **Ripristina**: vedi Versioni e ripristino.
- **Una schermata non mostra più un pezzo.** Un form, una tabella o un endpoint è stato tolto o rinominato. La pagina **Integrità del low-code** elenca i collegamenti rotti e propone il rimedio.

Parole da sapere

- **Record.** Un archivio del catalogo, con i suoi campi.
- **Elenco chiuso.** Un insieme di valori ammessi, ciascuno con codice ed etichetta.
- **Form.** La maschera dove si scrive.
- **Endpoint.** Una domanda ai dati con un nome: una lettura o una scrittura.
- **Parametro.** Un valore che un endpoint o una schermata riceve da fuori.
- **Flusso.** Il disegno che dice cosa succede e in che ordine.
- **Nodo.** Un passo del flusso.
- **Trigger.** Quello che fa partire un flusso da solo: un orario o un webhook.
- **Vista.** La schermata che l'utente apre, fatta di più pezzi.
- **Variabile.** Un valore con un nome che un form o una tabella conosce.

> **Per chi sviluppa.** La pagina dell'integrità è `/ade-next/integrita`. Cosa il motore sa fare e cosa no, riga per riga: `docs/low-code/capabilities.md`.