# Step 1: Creating the Initial App

giovedì 12 febbraio 2026 09:26

## Analisi dell'Infrastruttura Iniziale

Abbiamo scaricato e avviato un progetto basato su un template standard. Ecco gli elementi tecnici chiave che costituiscono le fondamenta dell'applicazione:

1. **Il Floorplan "Worklist":** L'architettura è progettata per gestire una collezione di oggetti.
    - **Master View (Worklist.view.xml):** È la schermata principale. Contiene una tabella (sap.m.Table) che elenca i prodotti. Include funzionalità standard come la ricerca e l'indicazione del numero di elementi.
    - **Object View:** Selezionando una riga della tabella, l'utente naviga verso una vista di dettaglio.
    - **Navigazione:** La logica di routing è già configurata nel manifest.json per gestire il passaggio tra la lista e il dettaglio.
2. **Gestione dei Dati (Mock Server):** Similmente al tutorial precedente, l'app utilizza un **Mock Server OData V2** (webapp/localService/mockserver.js).
    - Il file metadata.xml definisce lo schema dei dati (Entità "Products", proprietà come prezzo, stock, ecc.).
    - Al momento, il server genera dati casuali basati su questo schema, permettendo di testare l'interfaccia senza un backend reale.
3. **Configurazione (manifest.json):** Questo file centralizza la configurazione:
    - **sap.app**: Definisce la sorgente dati (mainService) che punta all'URI intercettato dal Mock Server.
    - **sap.ui5**: Gestisce i modelli (il modello OData di default e il modello i18n per le traduzioni) e imposta la rootView.
4. **Internazionalizzazione (i18n):** Come indicato nella nota del tutorial, i testi nel file webapp/i18n/i18n.properties sono stati generati automaticamente dal template.
    - **Azione consigliata:** È necessario revisionare questi testi. Ad esempio, stringhe come *"Enter an <Products> name"* devono essere corrette in un formato leggibile per l'utente finale (es. *"Cerca per nome prodotto"*).

# Step 2: Custom Mock Data

giovedì 12 febbraio 2026 09:30

## Procedura Operativa

Per implementare i dati personalizzati (Custom Mock Data), è necessario agire sulla struttura dei file del progetto seguendo questi passaggi:

1. **Predisposizione della Directory:** All'interno della cartella webapp/localService, deve essere creata una nuova sottocartella specifica per ospitare i file dati. Questa cartella funge da repository per le risposte simulate dal server.
2. **Creazione del Dataset "Prodotti":** È necessario creare un nuovo file JSON dedicato all'entità dei prodotti. Il nome di questo file è fondamentale: deve corrispondere esattamente al nome dell' *EntitySet* definito nel file dei metadati del servizio OData (metadata.xml). Il contenuto di questo file sarà un array di oggetti che rappresentano l'inventario, includendo proprietà quali identificativo, nome del prodotto, unità in magazzino, prezzo unitario e riferimento al fornitore.
3. **Creazione del Dataset "Fornitori":** Analogamente a quanto fatto per i prodotti, è necessario creare un secondo file JSON per l'entità dei fornitori. Anche in questo caso, la denominazione del file deve rispecchiare l'EntitySet corrispondente. Il contenuto includerà i dettagli anagrafici dei fornitori, come nome dell'azienda, indirizzo e contatti.

## Logica Tecnica

Il funzionamento del Mock Server in questa configurazione si basa su un meccanismo di **rilevamento automatico per convenzione**:

- Il Mock Server legge il file metadata.xml per comprendere la struttura del servizio.
- Quando riceve una richiesta di lettura (es. per ottenere la lista dei prodotti), verifica se nella cartella dei dati simulati esiste un file JSON con lo stesso nome dell'entità richiesta.
- **Se il file esiste:** Il server legge il contenuto statico del file e lo restituisce all'applicazione.
- **Se il file non esiste:** Il server genera dati casuali rispettando i tipi di dato definiti nei metadati (come avveniva nel Passaggio 1).

# Step 3: Extending the Worklist Table

giovedì 12 febbraio 2026 09:34

## 1. Ottimizzazione del Recupero Dati (Espansione)

Il primo requisito è mostrare il nome del fornitore per ogni prodotto. Nel modello dati, il "Prodotto" e il "Fornitore" sono due entità distinte collegate da una relazione.

- **Problema:** Se caricassimo solo i prodotti, l'app dovrebbe fare una chiamata aggiuntiva al server per ogni riga della tabella per scoprire il nome del fornitore. Questo sarebbe inefficiente.
- **Soluzione:** Configuriamo la tabella affinché richieda al server di "espandere" la relazione col fornitore. In pratica, chiediamo al server OData: *"Dammi tutti i prodotti e, già che ci sei, includi i dettagli del fornitore per ciascuno"*. Questo riduce il traffico di rete a una singola chiamata.

## 2. Definizione delle Colonne e Responsività

Dobbiamo aggiungere nuove colonne alla tabella per mostrare: Fornitore, Prezzo, Unità Ordinate e Unità in Magazzino. Tuttavia, lo spazio su uno schermo mobile è limitato.

- **Strategia:** Utilizziamo le proprietà di responsività della tabella SAPUI5.
    - **Colonne Prioritarie:** Il "Nome Prodotto" e le "Unità in Magazzino" sono essenziali, quindi rimarranno sempre visibili.
    - **Colonne Secondarie:** Il "Fornitore", il "Prezzo" e le "Unità Ordinate" vengono configurati per comportarsi dinamicamente. Su tablet e desktop saranno normali colonne; su smartphone verranno nascoste o spostate in un'area di dettaglio sotto la riga principale (funzionalità chiamata "Popin").

## 3. Logica Semantica (Stati del Magazzino)

Vogliamo che l'utente identifichi a colpo d'occhio le criticità. I numeri da soli non bastano; serve il colore.

- **Implementazione:** Creiamo una nuova funzione di formattazione (Formatter) che analizza la quantità in magazzino e restituisce uno "Stato" semantico:
    - **Errore (Rosso):** Se la quantità è 0 (Prodotto esaurito).
    - **Attenzione (Arancione):** Se la quantità è 10 o inferiore (Scorta bassa).
    - **Successo (Verde):** Se la quantità è superiore a 10 (Disponibilità adeguata). Questa logica viene applicata al numero visualizzato nella colonna "Unità in Magazzino".

## 4. Mappatura dei Dati

Infine, colleghiamo (bind) le celle della tabella ai dati specifici:

- La colonna Fornitore viene collegata al nome dell'azienda (reso disponibile dal passaggio 1).
- La colonna Prezzo viene formattata con la valuta (EUR).
- La colonna Unità in Magazzino viene collegata al numero reale e alla logica dei colori definita nel passaggio 3.

## 5. Etichette (i18n)

Aggiorniamo il file delle risorse linguistiche per assicurare che ogni nuova colonna abbia un'intestazione corretta e traducibile (es. "Fornitore", "Prezzo", ecc.).

# Step 4: Quick Filter for the Worklist

giovedì 12 febbraio 2026 09:40

## 1. Modifica dell'Interfaccia Utente (Worklist.view.xml)

Dobbiamo inserire una nuova barra di controllo sopra la tabella dei prodotti.

- **Strumento:** Utilizziamo un componente specifico per le schede con icone (IconTabBar).
- **Posizionamento:** Questo componente viene collocato nell'intestazione della pagina, subito prima della tabella esistente.
- **Struttura:** Definiamo quattro schede (Tabs), ognuna rappresentante uno stato specifico:
  1. **Tutti i prodotti:** Mostra il conteggio totale.
  2. **Disponibili (Plenty in Stock):** Prodotti con scorte adeguate (> 10). Usiamo un'icona verde (Positive).
  3. **In esaurimento (Shortage):** Prodotti con scorte basse (1-10). Usiamo un'icona arancione (Critical).
  4. **Esauriti (Out of Stock):** Prodotti con scorte a zero. Usiamo un'icona rossa (Negative).
- **Binding:** Ogni scheda è configurata per mostrare un contatore dinamico (count), che verrà calcolato dal controller.

## 2. Logica di Inizializzazione (Worklist.controller.js)

All'avvio del controller (onInit), dobbiamo preparare il terreno per il filtraggio:

- **Modello di Vista:** Estendiamo il modello locale (worklistView) aggiungendo le proprietà per i quattro contatori (inizializzati a 0).
- **Definizione dei Filtri:** Creiamo un oggetto interno che mappa le chiavi delle schede ai rispettivi filtri OData:
  - *Shortage:* Filtra dove UnitsInStock è tra 1 e 10.
  - *Out of Stock:* Filtra dove UnitsInStock è minore o uguale a 0.
  - *In Stock:* Filtra dove UnitsInStock è maggiore di 10.
  - *All:* Nessun filtro. Questa mappatura preventiva rende il codice molto pulito: quando l'utente clicca una scheda, sapremo già quale filtro applicare senza dover ricreare la logica ogni volta.

## 3. Calcolo dei Contatori (onUpdateFinished)

Per mostrare i numeri corretti sulle schede (es. "3 prodotti esauriti"), dobbiamo interrogare il server (o il Mock Server).

- **Trigger:** Sfruttiamo l'evento che segnala il completamento dell'aggiornamento della tabella.
- **Azione:** Eseguiamo tre chiamate di lettura separate al server, chiedendo specificamente il **conteggio ($count)** dei prodotti che soddisfano ciascuno dei filtri definiti al punto 2.
- **Aggiornamento:** Quando il server risponde, aggiorniamo le proprietà del modello di vista collegate alle schede. Grazie al data binding, i numeri nell'interfaccia si aggiorneranno automaticamente.

## 4. Gestione dell'Evento di Filtraggio (onQuickFilter)

Quando l'utente clicca su una delle schede, scatta un evento che dobbiamo gestire:

- **Identificazione:** Recuperiamo la "chiave" della scheda selezionata (es. "shortage").
- **Applicazione:** Usiamo questa chiave per recuperare il filtro pre-configurato (dal punto 2) e lo applichiamo al *binding* della tabella.
- **Risultato:** La tabella si aggiorna istantaneamente mostrando solo le righe che soddisfano il criterio scelto.

## 5. Testi (i18n.properties)

Infine, aggiungiamo le etichette testuali per le nuove schede nel file delle traduzioni, garantendo che l'interfaccia sia localizzata correttamente.

# Step 5: Adding Actions to the Worklist

giovedì 12 febbraio 2026 09:40

## 1. Selezione Multipla nella Tabella (Worklist.view.xml)

Per permettere azioni su più prodotti contemporaneamente, modifichiamo il comportamento della tabella.

- **Modalità:** Cambiamo la modalità di selezione in "MultiSelect".
- **Risultato Visivo:** Ogni riga della tabella avrà ora una casella di spunta (checkbox) all'inizio, permettendo all'utente di selezionare uno, nessuno o tutti i prodotti visibili.

## 2. Aggiunta dei Pulsanti di Azione (Worklist.view.xml)

Arricchiamo l'interfaccia aggiungendo due pulsanti nella barra a piè di pagina (Footer) della pagina semantica.

- **Pulsante "Ordina" (Positive Action):** Destinato all'incremento delle scorte. Sarà evidenziato semanticamente come azione positiva.
- **Pulsante "Rimuovi" (Negative Action):** Destinato all'eliminazione del prodotto. Sarà evidenziato semanticamente come azione negativa (o distruttiva).
- **Posizionamento:** Grazie all'uso della SemanticPage, questi pulsanti verranno posizionati automaticamente secondo le linee guida SAP Fiori, senza bisogno di CSS manuale.

## 3. Logica di Business nel Controller (Worklist.controller.js)

Nel controller, implementiamo le funzioni che rispondono al click sui nuovi pulsanti.

### A. Gestione dell'Azione "Ordina" (onUpdateStockObjects)

1. **Recupero Selezione:** Il codice verifica quali righe sono state spuntate dall'utente.
2. **Validazione:** Se nessun prodotto è selezionato, viene mostrato un messaggio di errore che invita l'utente a selezionare almeno un elemento.
3. **Ciclo di Aggiornamento:** Per ogni prodotto selezionato:
    - Recupera l'oggetto dati corrente.
    - Incrementa la proprietà "Unità in Magazzino" di un valore fisso (es. +10).
    - Invia una richiesta di **aggiornamento (UPDATE)** al modello OData.
4. **Feedback:** Una volta completate le operazioni, mostra un messaggio temporaneo ("Toast") che conferma l'aggiornamento delle scorte.
5. **Aggiornamento UI:** Poiché il modello OData è bidirezionale, la tabella si aggiornerà automaticamente mostrando i nuovi valori, e le righe potrebbero persino cambiare colore (es. da rosso a verde) grazie al formattatore implementato nei passaggi precedenti.

### B. Gestione dell'Azione "Rimuovi" (onUnlistObjects)

1. **Recupero Selezione:** Identifica i prodotti selezionati.
2. **Validazione:** Controlla che ci sia una selezione.
3. **Ciclo di Cancellazione:** Per ogni prodotto selezionato:
    - Recupera il percorso specifico dell'entità.
    - Invia una richiesta di **cancellazione (DELETE)** al modello OData.
4. **Feedback:** Mostra un messaggio di conferma ("Prodotto rimosso").
5. **Aggiornamento UI:** Le righe corrispondenti spariranno immediatamente dalla tabella.

### C. Gestione degli Errori e Successi

Per entrambe le azioni, implementiamo delle funzioni di callback (successo/errore). Nel tutorial, per semplicità, assumiamo che le chiamate al Mock Server vadano sempre a buon fine e mostriamo il messaggio di successo solo quando l'ultima richiesta del lotto è stata processata.

## 4. Testi (i18n.properties)

Aggiungiamo le etichette per i nuovi pulsanti ("Ordina", "Rimuovi") e i messaggi di feedback ("Nessun prodotto selezionato", "Scorte aggiornate", ecc.) nel file delle risorse linguistiche.

# Step 6: Extending the Detail Page

giovedì 12 febbraio 2026 09:42

## 1. Arricchimento della Testata (Header)

La parte superiore della pagina (l'intestazione semantica) viene potenziata per mostrare a colpo d'occhio le informazioni critiche. Non ci limitiamo più al solo titolo del prodotto, ma introduciamo un layout diviso in due aree principali tramite un contenitore flessibile (FlexBox):

- **Attributi Chiave (Sinistra):** Vengono aggiunti campi testuali per mostrare l'**ID Prodotto** e il **Prezzo Unitario**. Questi utilizzano i formattatori standard per garantire che numeri e valute siano visualizzati correttamente (es. "15,00 EUR").
- **Stato del Magazzino (Destra):** Questa sezione è dedicata alla visualizzazione immediata della disponibilità:
    - **Numero Unità:** Mostra la quantità numerica esatta.
    - **Indicatore di Progresso (Barra):** Introduciamo un elemento visivo (ProgressIndicator) che mostra una barra colorata proporzionale alla quantità in magazzino. La cosa fondamentale è che **riutilizziamo lo stesso formattatore** creato per la tabella: la barra sarà verde, arancione o rossa a seconda delle soglie (>10, <=10, 0). Questo garantisce coerenza visiva tra la lista e il dettaglio.
    - **Stato "Fuori Produzione":** Un indicatore di stato (ObjectStatus) che appare *solo* se il prodotto è contrassegnato come "Discontinued". Utilizziamo la proprietà di visibilità dinamica per nasconderlo se il prodotto è ancora attivo.

## 2. Sezione Contenuto: Informazioni Fornitore

Sotto l'intestazione, nell'area principale della pagina, aggiungiamo un nuovo pannello dedicato al fornitore.

- **Pannello (Panel):** Crea un'area distinta visivamente con il titolo "Informazioni Fornitore".
- **Modulo Semplice (SimpleForm):** All'interno del pannello, utilizziamo un controllo "Form" per impaginare i dati. Il vantaggio del SimpleForm è che gestisce automaticamente l'allineamento tra etichette (es. "Indirizzo") e valori (es. "Via Roma 1"), adattandosi perfettamente sia a schermi desktop che mobile.
- **Dati Visualizzati:** Mostriamo Nome Azienda, Indirizzo, CAP, Città e Paese.
    - *Nota Tecnica:* Questi dati sono accessibili (Supplier/City, Supplier/Address) senza nuove chiamate al server perché nel passaggio 3 abbiamo impostato l'espansione automatica (expand) dei dati del fornitore quando si carica il prodotto.

## 3. Internazionalizzazione (i18n)

Come consuetudine, tutte le nuove etichette introdotte (come "Prezzo", "Fuori Produzione", "Nome Fornitore", "Indirizzo") vengono aggiunte al file delle risorse linguistiche per permettere la traduzione dell'app.

# Step 7: Adding a Comments Section

giovedì 12 febbraio 2026 10:19

## 1. Creazione e Registrazione del Modello Commenti

Non avendo un backend reale per salvare i commenti, creiamo un modello JSON locale che fungerà da "database volatile" per la durata della sessione.

- **Definizione del Modello (models.js):** Aggiungiamo una funzione createCommentsModel che restituisce un nuovo JSONModel inizializzato con un array vuoto productComments.
- **Istanziazione Globale (Component.js):** Nel file principale del componente, invochiamo questa funzione e registriamo il modello con il nome **"productFeedback"**. Rendendolo globale, sarà accessibile da qualsiasi vista o controller dell'app.

## 2. Modifica della Vista di Dettaglio (Object.view.xml)

Estendiamo la pagina di dettaglio per ospitare la sezione commenti.

- **Layout:** Aggiungiamo un nuovo Panel sotto le informazioni del fornitore. Poiché la SemanticPage accetta un solo controllo nella sua aggregazione content, dobbiamo avvolgere i due pannelli (Fornitore e Commenti) in un VerticalLayout.
- **Input Commenti:** Utilizziamo il controllo sap.m.FeedInput. Questo fornisce un campo di testo e un pulsante "Invia" pronti all'uso.
- **Lista Commenti:** Sotto l'input, aggiungiamo una List per mostrare i commenti esistenti.
    - **Binding:** La lista è collegata al path /productComments del modello productFeedback.
    - **Template:** Usiamo FeedListItem per visualizzare ogni commento in uno stile "social" (icona, testo, data).
    - **Ordinamento:** Configuriamo il binding per ordinare i commenti per data in modo decrescente (dal più recente).

## 3. Logica del Controller (Object.controller.js)

Il controller deve gestire due aspetti: salvare i nuovi commenti e filtrare quelli visualizzati.

**A. Salvataggio del Commento (onPost)**

Quando l'utente invia un commento:

1. Il controller recupera il testo inserito e l'ID del prodotto corrente.
2. Crea un oggetto dati con: ID prodotto, tipo ("Comment"), data corrente formattata e il testo del commento.
3. Accede al modello productFeedback, recupera l'array dei commenti e vi aggiunge il nuovo oggetto.
4. Aggiorna il modello, causando il refresh automatico della lista nell'interfaccia.

**B. Filtraggio dei Commenti (_onBindingChange)**

Poiché tutti i commenti di tutti i prodotti finiscono nello stesso array del modello globale, dobbiamo assicurarci che la pagina di dettaglio mostri solo quelli pertinenti al prodotto visualizzato.

- Ogni volta che cambia il prodotto visualizzato (evento di binding), applichiamo un **filtro** alla lista dei commenti: productID EQ [ID Corrente].

## 4. Testi (i18n.properties)

Aggiungiamo le etichette per il titolo della sezione ("Commenti") e per il messaggio di lista vuota ("Nessun commento").

## Risultato Finale e Conclusione

Ora, aprendo un prodotto, puoi scrivere un commento e vederlo apparire immediatamente nella lista sottostante. Se cambi prodotto, i commenti spariranno (filtrati), ma tornando al primo prodotto li ritroverai (finché non ricarichi la pagina).

**Riepilogo del Tutorial:** Hai costruito un'applicazione completa partendo da un template. Hai imparato a:

1. Configurare un Mock Server con dati realistici.
2. Estendere la tabella con nuove colonne e formattazione condizionale.
3. Implementare filtri rapidi e azioni di massa (Ordina/Rimuovi).
4. Arricchire la vista di dettaglio con layout complessi e modelli dati multipli.